CHIP PLUS

EINE BEILAGE
IN CHIP NR. 9
SEPTEMBER 1989

80386-ARCHITEKTUR

CHIP WISSEN

Die kompetente Buchreihe rund um den PC



Honerkamp/Jetter:

Fliegen mit dem Flight Simulator III

236 Seiten, zahlr. Bilder
Harteinband, Fadenheftung
40,– DM/ISBN 3-8023-**0274**-5

Das Buch beschreibt ausführlich das Flieger-Einmaleins, das zum Beherrschen des Simulationsprogramms **„Flight Simulator III“** notwendig ist. Ausführlich wird die Instrumentenfliegerei mit den dazugehörigen Karten behandelt. Einige Neuerungen, wie der Anflug auf San Francisco Area und Scenery Disk Europe, der Anflüge auf bekannte deutsche und europäische Flughäfen erlaubt, erhöhen den Spaß an diesem Buch.
Die Autoren wünschen schon heute „Hals- und Beinbruch“ und „happy landings“.



Hans-Ulrich Wolff:

Schneller erfolgreich mit Framework III

180 Seiten, 55 Bilder
Harteinband, Fadenheftung
38,– DM/ISBN 3-8023-**0287**-7

Das Buch beschreibt die deutsche Framework-III-Version. Es erleichtert den Einstieg und unterstützt den Anwender beim Bewältigen der vielfältigsten Aufgaben anhand leicht nachvollziehbarer Beispiele. Die Programmiersprache FRED wird einführend und ihre Funktion umfassend vorgestellt.

Framework III eignet sich hervorragend für die Bewältigung der typischen Aufgabenstellungen am MS-DOS-Rechner:
● konzeptionelles Arbeiten ● Textverarbeitung ● Tabellenkalkulation ● Datenbanken ● Grafik und Telekommunikation



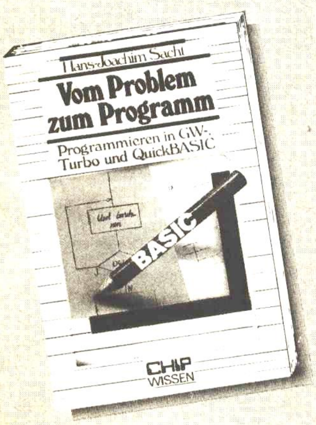
Hans-Joachim Sacht:

Programmieren mit QuickBASIC 4.0 und 4.5

210 Seiten, 66 Bilder, Harteinband
mit 5,25-Zoll-Diskette
48,– DM/ISBN 3-8023-**0245**-1

Durch die neuen Microsoft-Compiler QuickBASIC 4.0 und 4.5 können so komfortable und gutstrukturierte Programme entwickelt werden, wie es sonst nur mit einem leistungsfähigen Interpreter möglich war.

Das Buch bietet die vielfältigen Funktionen in konzentrierter Form und geht vor allem auf die Handhabung und Sprachelemente dieses Compilers ein. Programmierern, die bisher nur mit einem BASIC-Interpreter gearbeitet haben, wird erklärt, wie sie schnell auf QuickBASIC umsteigen und so die Vorteile der Kompilierung ausnutzen können. Eine Diskette mit den im Buch behandelten Beispielen liegt bei.



Hans-Joachim Sacht:

Vom Problem zum Programm

Programmieren in GW-BASIC, Turbo BASIC und QuickBASIC

464 Seiten, 183 Bilder
4., völlig überarbeitete Auflage 1988
48,– DM/ISBN 3-8023-**0214**-1

Das von Microsoft entwickelte GW-BASIC bzw. BASICA gilt inzwischen bei 16-bit-Computern als Standard und für 32-bit-Rechner als Grundlage des Sprachumfangs auch anderer neuer Dialekte.
Die Neukonzeption gliedert sich in 3 Teile:

1. Teil erklärt Grundlagen der BASIC-Programmierung unter Einbeziehung der wichtigsten MS-DOS-Kommandos, Compiler.
2. Teil bringt zahlreiche Programmroutinen für breite Anwendungen.
3. Teil zeigt, wie ein Programm Schritt für Schritt entsteht.

Eine Diskette mit den im Buch behandelten Beispielen ist erhältlich.

VOGEL

Haben Sie schon den neuen „CHIP- Katalog“?
Bestellen Sie gleich!

Vogel Buchverlag
Postfach 67 40 · 8700 Würzburg 1

Markt und Macht

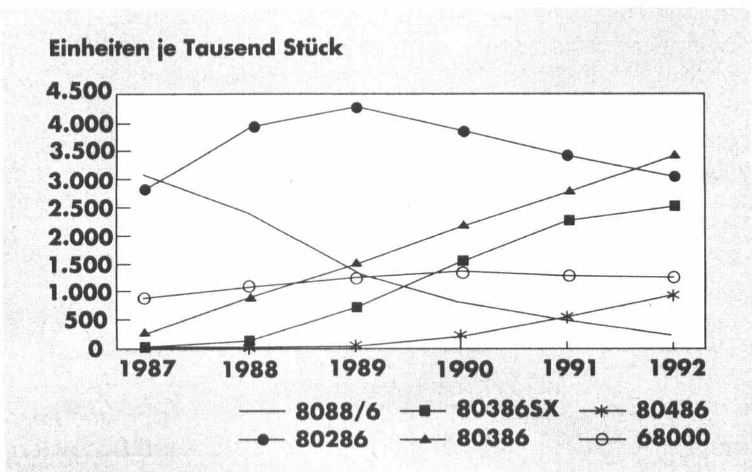
Ulrich Parthier

Trotz technischer Innovationen und vieler Vorzüge der in letzter Zeit viel diskutierten RISC-(Reduced Instruction Set Computer-)Prozessoren – in der PC-Welt dominieren nach wie vor Mikrochips, die auf CISC-(Complex Instruction Set Computer-)Konzepten basieren. Die bekanntesten Vertreter sind hier die Motorola-68000-Familie und die Intel-Prozessoren. Hier hat Intel mit der 80386-Architektur und seinem neuen Flaggschiff, dem i486, neue Maßstäbe gesetzt.

Ende 1985 schlug mit der Vorstellung des 80386-Prozessors Intels große Stunde, revolutionierte dieser Chip doch in seiner Architektur und Leistungsfähigkeit die PC-Welt. Mittlerweile sind fast vier Jahre vergangen, und allmählich wird der Traum, Mainframe-Power auf den Schreibtisch zu holen, zur Realität.

Wie bei allen Neuankündigungen, so werden die Hersteller auch diesmal Rechner mit dem 80486-Prozessor en masse erst ein Jahr nach der Ankündigung zeigen. Der PC mit dem 8086/88-Prozessor ist spätestens seit dem Preisrutsch für 386-Rechner in diesem Jahr „out", und, was die Preise angeht, mittlerweile schon fast im Home-Computer-Bereich angesiedelt. Standard ist heute ein AT-Rechner mit dem 80286-Prozessor. Erst 1992, so die Marktforscher, wird der 80386-Chip solche Rechner, die mit dem 80286 ausgerüstet sind, überholen. Bis dahin werden auch hier die Preise auf breiter Front ins Rutschen geraten.

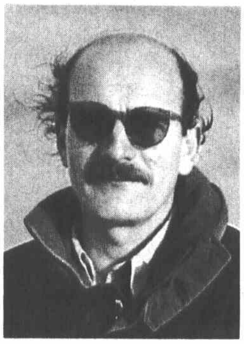
Der 80486 wird sicherlich noch mindestens fünf Jahre brauchen, bevor er zum Standard-Prozessor in Personal-Computern wird. Gut Ding will eben Weile haben, und ehrlich gesagt, bei allem Nutzen für die Schar der Software-Entwickler und trotz OS/2 und Unix: Welcher „normale" PC-Anwender braucht denn wirklich die 80486-Power? Womit gesagt sein soll, daß natürlich alle PC auf Basis der Intel-Prozessoren weiterhin ihre Existenzberechtigung haben. Schließlich hat Intel bei allem Fortschritt immer auf eines geachtet: Der Anwender kann „seine" MS-DOS-Programme weiterverwenden und gegebenenfalls mit der Prozessor-Power mitwachsen.



IDC-Marktvorhersage des Mikroprozessor-Absatzes

Grundsätzliche Design-Überlegungen

Dr. Jens Bodenkamp



Der Autor
Dr. Jens Bodenkamp studierte Physik in Hamburg und Karlsruhe. Über seine Arbeit am DESY (Deutsches Elektronensynchrotron) in Hamburg und an weiteren physikalischen Instituten, die sich mit Elementarteilchenphysik beschäftigten, kam er 1982 zu Intel nach München. Dort bekleidet er seit 1986 die Stellung des Produkt-Managers für Intels Mikroprozessor-Plattformen.

Seit Intel den 32-bit-Mikroprozessor 80386 im Oktober 1985 weltweit einführte, ist sehr viel geschehen. Inzwischen gibt es den 80386DX mit seinen Peripherie-Chips, den numerischen Koprozessor 80387 und den 82385-Cache-Controller in vier Geschwindigkeitsversionen (16, 20, 25 und 33 MHz), den 80386SX am unteren Leistungsende als preiswertesten Vertreter und seit neuestem den i486-Prozessor als das Flaggschiff der 386-Architektur.

Jedes Programm läuft Binärcode-kompatibel sowohl auf der langsamsten als auch auf der schnellsten Maschine. Die vier weltweit verbreiteten Standardbetriebssysteme Unix, DOS, OS/2 und Windows 386 unterstützen die Rechner der 386-Architektur. Alle Prozessoren dieser Architektur sind darüber hinaus in der Lage, einen oder mehrere 8086-Prozessoren zu emulieren, das heißt, zum Beispiel DOS-Anwendungsprogramme unter dem 32-bit-Betriebssystem Unix auf ein und demselben Prozessor quasi gleichzeitig zu verarbeiten.

Rechnerarchitektur: Was ist das?

Welche Abstraktionsebene auch immer zur Formulierung einer Aufgabe benutzt wird, am Ende steht immer ein Stück Maschinencode, das als die letzte Abbildung meines Problems angesehen werden kann. War ich in der glücklichen Lage, mich in der Sprache der vierten Generation auszudrücken, wird diese Ebene in einem nächsten Schritt auf einem niedrigeren Abstraktionsniveau abgebildet werden, wahrscheinlich auf eine Hochsprache wie C. Der C-Compiler wird dann entweder direkt den gewünschten Maschinencode erzeugen oder eine weitere Abbildung auf den Assembler des Rechners vornehmen, der seinerseits den letzten Abbildungsschritt vollzieht. Von einem einmal erzeugten Maschinencode werden weitere interne Abbildungsschritte über den Mikrocode eines Prozessors bis zum Schalten von Transistoren ausgeführt. Abstrakt läßt sich der letzte Vorgang in Boolescher Algebra formulieren.

Bis also die einmal formulierte/programmierte Aufgabe kommuniziert und wirklich durch die Transistoren eines Prozessors ausgeführt wird, durchläuft sie viele Schritte der Übersetzung bzw. Abbildung. Der Grund dafür ist sehr einfach einzusehen. Man möchte möglichst allgemein, bequem und automatisch von einer problemnahen und problemangepaßten formalen Beschreibung zu einer schaltkreisnahen Beschreibung in Boolescher Algebra kommen. Dem menschlichen Wesen am nächsten wäre eine natürlichsprachliche Formulierung einer Aufgabe und Eingabe in einen Rechner, der seinerseits in vielen Schritten für die Formulierung in der Sprache sorgt, die ihm wiederum am nächsten kommt, nämlich in Boolescher Algebra.

Übersetzer bilden also eine Sprache auf die andere ab. Alle üblichen Rechner können als „universale Turing-Maschinen" (benannt nach dem englischen Informatiker A. M. Turing) angesehen werden, die jede andere Turing-Maschine darstellen können. Zusammen mit einem Übersetzer stellt ein Rechner mit seinem Betriebssystem also eine weitere universale Turing-Maschine dar. Allein die Ökonomie der Problemlösung bestimmt, wie viele und welche Abbildungen vorgenommen werden müssen, um die Aufgabe von Transistorschaltern lösen zu lassen.

Genau eine der verschiedenen Ebenen der Abbildungen von Programmgenerationen über grafische Benutzerschnittstellen bis zum Transistorschaltkreis wird vom Hersteller des Rechners als Architektur festgelegt. Alles, was oberhalb der Architektur liegt, bezeichnet man als „Software", alles, was unterhalb liegt, als „Hardware". Per definitionem und durchaus beliebig ist die Architektur der Schnittstelle zwischen Hardware und Software. So kann es durchaus passieren, daß eine unglückliche Wahl der Architektur zu einem technisch und kommerziellen Mißerfolg führt.

Die Festlegung der Rechnerarchitektur folgt ganz analogen Gesetzen. Auch hier gibt die Gemeinde der potentiellen Benutzer Zielvorgaben für Funktionalität,

Preis und Rechenleistung, während der Rechnerarchitekt die Aufgabe erfüllt, innerhalb dieses Rahmens zu einer möglichst eleganten, effektiven und zukunftsweisenden Lösung zu kommen. Liegt die Architektur eines Rechners einmal fest und hatte sie Erfolg, so entsteht eine gewisse Infrastruktur. Es gibt dann eine Vielzahl von Übersetzern, Werkzeugen, Anwendungs-Software, verschiedene Betriebssysteme, Benutzerschnittstellen und dergleichen für diese Architektur.

Gesetzmäßigkeiten

Eine gut eingeführte Architektur kann aber im Laufe der Zeit zu einem Hemmschuh der Hard- und Software-Entwicklung führen, wenn Fortschritte in Technik und Wissenschaft erzielt worden sind. Wenn die Einführung einer neuen Architektur geplant wird, ist der Übergang von der alten auf eine neue Architektur ein wesentlicher kaufmännischer und technischer Gesichtspunkt, um Investitionen an Geld, Arbeit und Wissen in die alte Architektur nicht verfallen zu lassen. Aus dieser Überlegung heraus entstanden die Konzepte von Emulation und Simulation. Laut vorläufiger Definition im ISO-TC-97-Papier ist die Emulation

1. die gesamte oder teilweise Nachahmung eines Rechners durch einen anderen, vorzugsweise durch Hardware, so daß der nachahmende Rechner dieselben Daten akzeptiert, dieselben Programme durchführt und dieselben Ergebnisse erbringt wie der nachgeahmte Rechner;
2. die Nutzung von Programmtechnik und Maschinenzusätzen, um einem Rechner zu erlauben, Programme durchzuführen, die für einen anderen geschrieben sind.

Simulation ist die Darstellung von Befehlsfolgen eines Rechners durch solche eines anderen Rechners. Eine bestimmte Maschine wird von einer anderen in Software nachgeahmt. Auch hierfür gibt es Beispiele in der Welt der PC und Workstations. Solche DOS-Simulations-Software wird etwa für den 68030 von Motorola angeboten.

Emulation und Simulation

Vereinfacht gesagt, dienen Emulation und Simulation zur Darstellung der Architektur X auf dem Rechner der Architektur Y. Bei der Emulation besitzt der Rechner Y Hardware-Einrichtungen zur Darstellung der Architektur X, bei der Simulation wird X durch Software auf Y dargestellt. Viele Zwischenformen sind möglich und in der Geschichte der Datenverarbeitung verwirklicht worden. Beide Methoden sind mit Leistungsverlusten verbunden. Die alten Programme laufen im Emulationsmodus stets langsamer, als wenn sie für die neue Architektur umgeschrieben werden würden. Die Verluste bei reiner Emulation liegen zwischen 10 Prozent und 50 Prozent, bei reiner Simulation über 90 Prozent bis weit über 99 Prozent. So werden DOS-Applikationen unter Unix auf einer Sun 386i etwa um 10 Prozent langsamer ausgeführt. Die ungünstigste Nachahmung ist Simulation von Maschinencode. In diesem Fall muß jeder alte Befehl (= aus der alten Architektur) vom Simulator geprüft und in Befehlsfolgen der neuen Architektur umgesetzt werden. Wenn die neuen Befehlsfolgen ein- bis fünfmal so lang sind wie die alten (0 Prozent bis 80 Prozent Leistungsverlust), so dürfte der Aufwand für die Erzeugung der neuen Befehlsfolgen in der Größenordnung von zehn bis 500 Befehlen pro alten Befehl liegen (90 Prozent bis 99,8 Prozent Leistungseinbußen).

Eine Maschinenarchitektur, die sich am Markt durchsetzen konnte, wird immer in evolutionären Schritten von Generation zu Generation weiterentwickelt. Eine neue Implementation kann etwa dem Benutzer eine erhebliche Steigerung der zur Verfügung stehenden Rechenleistung an die Hand geben. Der Instruktionssatz könnte einen oder mehrere neue und wichtige vom Benutzer geforderten Befehle enthalten usw. Die alten Programme laufen aber binär-kompatibel ohne jegliche Änderung auf den neuen Implementationen der Architektur, allerdings sehr viel schneller, ab.

Beispiele

Nun gibt es Maschinen mit sehr hoch liegenden Architekturen. Beispiele sind Datenbank- und dedizierte Lisp-Rechner. Eine Maschine mit extrem tief liegender Architektur stellen RISC-Computer (Reduced Instruction Set Computer) dar.

Es bleibt festzuhalten, daß das RISC-Konzept die Architekturlinie zwischen Hardware und Software in Richtung Software verschiebt. Auf der anderen Seite ist man in der Lage, die Fortschritte im Compiler-Bau näher an die Schaltkreise heranzutragen. Architekturen für Universalrechner hingegen liegen in ihrer Komplexität auf einer mittleren Ebene.

Die PC-Welt wird revolutioniert

Dr. Jens Bodenkamp

Auf unterster Ebene unterstützt die 386-Architektur bestimmte Datentypen, wie Byte, ganze Zahlen mit und ohne Vorzeichen, Bit usw. Sie besitzt einen mächtigen Befehlsvorrat von etwa 300 Instruktionen, bei dem die Befehle der numerischen Architektur zur Unterstützung von Gleitkommaarithmetik mitgezählt wurden. Es wurden bestimmte Adressierungsarten festgelegt, ferner ein Speichermodell auf niedrigster Ebene und ein Registermodell. Darüber hinaus definiert die Hardware der 386-Architektur Speicherverwaltung und Speicherschutz, ein Interrupt-Konzept und die Behandlung von Tasks.

Datentypen

Die Implementationen der 386-Architektur sind Byte-orientierte Maschinen. Zwei Byte werden Wort und vier Byte Doppelwort genannt. Das niederwertigste Byte eines Wortes oder Doppelwortes wird stets an der numerisch kleinsten Adresse im Speicher abgelegt. Man nennt diese Art „Little Endian". Das umgekehrte Verfahren wird als „Big Endian" bezeichnet. Technisch ist das eine so gut wie das andere. Ganze Zahlen mit (Integer) und ohne Vorzeichen (Ordinale) werden als Byte- (8-bit*), Wort- (16-bit*) oder Doppelworttyp (32-bit) unterstützt. In den gleichen Quantitäten sind Zeichenketten festgelegt worden. So gibt es Byte-, Wort- und Doppelwort-Zeichenketten, auf die bestimmte Instruktionen wirken können.

Ein wichtiger Datentyp insbesondere in Grafik- und Kommunikationsanwendungen ist das Bit. Die 386-Architektur unterstützt diesen Datentyp in einem direkt adressierbaren ±2 Gigabit großen Bereich. Ein Bit-Feld ist 32 bit lang. Weitere Datentypen sind „Binary Coded Decimal" (BCD) mit einer Ziffer pro 8 bit und gepackten BCD-Zahlen mit zwei Ziffern pro Byte. Es werden darüber hinaus 32-bit (Near Pointer) und 48-bit lange Zeiger (Far Pointer) erkannt und behandelt. Ein Near Pointer enthält eine logische Adresse, welche auf einer Speicherzelle innerhalb eines bis zu 4 Gigabyte großen Segmentes zeigt. Dieser Zeigertyp wird benutzt, wenn sich der Systemprogrammierer für ein lineares und flaches Speichermodell entscheidet. Wird eine segmentierte Organisation gewählt, benutzt man den logischen Inhalt des Far Pointers, der aus einem 16 bit langen Auswahlindex (Segmentselektor) des zu adressierenden Speichersegments und einem 32 bit langen Offset innerhalb des ausgewählten Segments besteht.

Der Instruktionssatz

Der Instruktionssatz der 386-Architektur enthält den Befehlsvorrat, mit dessen Hilfe Daten und Adressen manipuliert oder getestet werden können. Man unterteilt ihn in den sogenannten Kern- und in den erweiterten Instruktionssatz. Zu den Kernbefehlen gehören alle die, welche in Anwendungs- und Systemprogrammen im Durchschnitt am häufigsten vorkommen. Dabei handelt es sich um Register-Register-Befehle, Load- und Store-Operationen sowie bedingte und unbedingte Verzweigungen. Diese werden erweitert durch Instruktionen, die solche Zeichenketten manipulieren, die mit dem Speicherschutz und der Speicherverwaltung zusammenhängen, ferner Prozessor-Kontrollbefehle, außerdem solche, die iterative Aktionen der Maschine steuern können, und schließlich solche, die blockorientierte Hochsprachen unterstützen. Bei der Implementation der Mikroarchitektur wird man darauf hinarbeiten, daß die Kerninstruktionen möglichst schnell ausgeführt werden. Da die Befehle des erweiterten Instruktionssatzes in Anwendungs- und Systemprogrammen nicht so häufig vorkommen, sondern z.B. nur zur Laufzeitunterstützung von Übersetzern (Compilern) oder zur Realisierung bestimmter Betriebssystem-Funktionen, darf deren Ausführung etwas länger dauern.

Der Instruktionssatz ist in neun Klassen eingeteilt, wobei jede Kategorie eine andere Operation unterstützt:
- Datentransfer
- Arithmetik
- Shift und Rotate

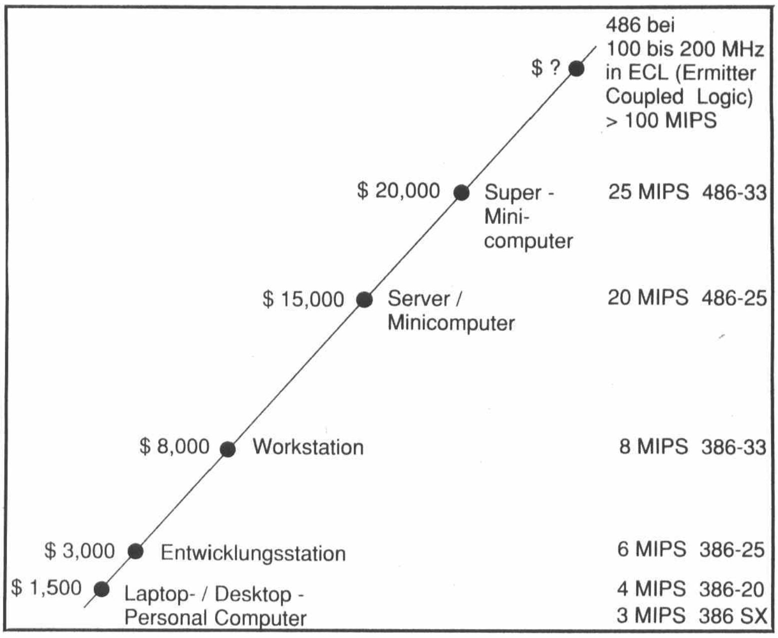
- Zeichenketten-Manipulation
- Bit-Manipulation
- Befehle zur Programmsteuerung
- Instruktionen zur Unterstützung von Hochsprachen
- Befehle zur Unterstützung von Betriebssystem-Funktionen und
- zur Steuerung des Prozessors.

Die Instruktionen können auf null, eins, zwei oder drei Operanden wirken. Der Operand kann sich dabei in einem Register, in der Instruktion selbst oder im Speicher befinden. Befehle, die keinen Operanden benötigen, wie STI (Set Interrupt Enable), sind meistens nur ein Byte lang. Instruktionen mit nur einem Operanden besitzen im allgemeinen eine Länge von zwei Byte. Im Durchschnitt, über eine Vielzahl von Anwendungs- und Systemprogrammen (über 31 Millionen Instruktionen) gemessen, beträgt die mittlere Befehlslänge 3,2 Byte. Die 386-Architektur unterstützt 8, 16 und 32 bit breite Operanden. Da sie gegenüber den 16-bit-Architekturen (8086- und 80286-Prozessor-Implementationen) aufwärts-Objektcodekompatibel ist, kann sie sowohl 16-bit- als auch 32-bit-Programme ausführen. Sogar Mischungen von 16-bit- und 32-bit-Programmteilen sind möglich.

Ein Bit entscheidet, um welchen Codetyp es sich handelt. Ein 16-bit-Codesegment unterstützt 8 und 16 bit breite Operanden. Ein spezielles Prefix-Byte, der Instruktion vorangestellt, überschreibt die voreingestellte Operandenlänge, so daß in einem solchen 16-bit-Codestück auch 32-bit-Operanden verwendet werden können. Umgekehrt erwartet ein 32-bit-Programmteil 8 oder 32 bit breite Operanden. Sollen solche einer Breite von 16 bit manipuliert werden, was im übrigen äußerst selten vorkommt, wie Analysen auf einer VAX gezeigt haben, muß man ebenfalls mit einem Byte-Prefix arbeiten, um die Voreinstellung zu überschreiben.

Adressierungsarten

Die 386-Architektur stellt Instruktionen von elf Adressierungsarten zur Verfügung, um Operanden zu adressieren. Diese wurden so gewählt, daß Hochsprachen wie C, Fortran usw. möglichst wirkungsvoll unterstützt werden können. Es ist ein Unterschied, ob man eine Konstante im Speicher adressieren möchte, das Element eines Vektors ist, oder eine Struktur adressieren will, die das Element eines Feldes ist. Für alle diese Möglichkeiten gibt es eine Adressierungsart, die der Compiler-Bauer ausnutzen kann, um möglichst bequem und angemessen die Operandenadressen zu bilden. Die effektive Adresse wird wie folgt gebildet:
Effektive Adresse = [Basisregister] + [Indexregister [* Skalierungsfaktor]] + [Displacement]



Bandbreite der Intel 386-Architektur

Die eckigen Klammern bedeuten, daß der jeweilige Term optional ist. Der Skalierungsfaktor kann 1, 2, 4 oder 8 betragen. Als Basisregister kann jedes der allgemeinen Register, das die 386-Architektur zur Verfügung stellt, benutzt werden. Gleiches gilt für das Indexregister mit der Ausnahme, daß das Stackpointer-Register ESP nicht verwendet werden darf. Dies stellt jedoch keine Einschränkung dar, da es kaum sinnvolle Programme gibt, die keinen Stack benutzen. Das Displacement, das der Instruktion folgt, kann 8 oder 32 bit lang sein.

Es wurde bereits darauf hingewiesen, daß aus Kompatibilitätsgründen Mischungen von 16-bit- und 32-bit-Programmteilen möglich sind. Ein Bit im Deskriptor des Codesegmentes – das D-Bit – entscheidet über die Voreinstellung (Deskriptoren und die Bedeutung von Segmenten werden später diskutiert). Ist D=0, nimmt man alle effektiven Adressen und Operandenlängen als 16 bit lang an und als 32 bit, wenn D=1 ist. Ein Beispiel: Es ist die Assembler-Anweisung

MOV DX, TABELLE [ESI * 2]

innerhalb eines Codesegmentes mit D=0 programmiert worden. Der Assembler

wird automatisch für diesen Befehl ein Adreßlängen-Prefix erzeugen, um die Adressierung im 32-bit-(physikalischen)-Speicherraum zu erlauben. Das folgende Beispiel führt zur Bildung eines Operandenlängen-Prefixes, wenn D = 1 ist:

MOV MEM16, DX

Auch dies wird vom Assembler automatisch erzeugt. Beide Prefixe können separat, aber auch zusammen auftreten.

Das Registermodell

Die 386-Architektur verfügt über sieben Registrierkategorien. Das sind:

- die allgemeinen Register
- die Segmentregister
- das Programmzeiger- und Flag-Register
- die Kontrollregister
- die System-Adressierungsregister
- die sogenannten Debug-Register
- die Testregister

Es gibt acht allgemeine 32 bit breite Register. Sie enthalten entweder Adressen oder Operanden. Sowohl die Adressierungsarten als auch die arithmetischen und logischen Verknüpfungen von Operanden mittels der Instruktionen aus dem Kernbefehlssatz sind nicht registerspezifisch. Das heißt beispielsweise, daß der Inhalt eines beliebigen Registers zu dem irgendeines anderen addiert werden kann. Auf die unteren 16 bit eines Registers kann aus Kompatibilitätsgründen separat zugegriffen werden. Die Register AX, BX, CX und DX teilen sich in einen 8 bit breiten höherwertigen und einem niederwertigen Teil, die ebenfalls einen individuellen Zugriff erlauben. Auch diese Möglichkeit wurde aus Kompatibilitätsgründen zum 8086 vorgesehen.

Das EIP-Register enthält immer den 32 bit breiten Offset der nächsten Instruktion, die ausgeführt werden soll. Dieser Offset ist immer relativ zur Basisadresse des Codesegmentes. Bei 16-bit-Adressierung werden nur die unteren 16 bit berücksichtigt (IP).

Die Segment- mit den dazugehörigen Deskriptorregistern sind 16 bit breit. Jedes einzelne Register enthält den Index auf das Segment im Speicher, auf das im Augenblick zugegriffen wird. Die Segmentlänge kann irgendwo zwischen einem Byte und 4 Gigabyte liegen. Maximal überdekken sie also den gesamten physikalischen und linearen Adreßraum, den die Architektur bereitstellt. CS bezeichnet das Segment für den Programmcode, SS das Segment für den Stack und die übrigen vier die Segmente für Daten. Sind die zu den Segmentregistern gehörigen Basisadressen mit 0 geladen und werden diese auch während des Programmablaufs nicht geändert, liegt das einfachste Speichermodell vor, das die 386-Architektur unterstützt. Man bezeichnet es als „flach" oder „linear", da ihm keinerlei Struktur unterlegt ist und alle Segmente bei der Adresse Null beginnend übereinanderliegen.

Mit jedem Segmentregister ist ein für den Programmierer unsichtbares Deskriptorregister assoziiert, das bei einem Speicherzugriff automatisch geladen wird. Logisch enthalten diese Register für jedes einzelne Segment dessen Basisadresse (ist im flachen Speichermodell immer null), seine Grenze (maximal 4 Gigabyte) und einige Attribute, die das Segment entsprechend qualifiziert. So können zum Schutz mehrerer Benutzer einer Maschine auf der Basis der 386-Architektur untereinander Segmente als nicht beschreibbar gekennzeichnet sein. Wird trotzdem versucht, schreibend auf ein solches zuzugreifen, erzeugt die Hardware der Maschine eine Ausnahmebedingung, an deren Ausgang das Betriebssystem geeignete Maßnahmen ergreift, um den Wiederholungsfall zu verhindern. Ähnliches gilt für die Segmentgrenze, die bei jedem Speicherzugriff auf dem Chip geprüft wird. Wurde sie überschritten, wird ebenfalls hardwaremäßig eine Ausnahmebedingung erzeugt, die in der Nomenklatur der 386-Architektur als „General Protection Fault" gekennzeichnet ist.

Die vier Prozessorkontroll-Register heißen CR0, CR1, CR2 und CR3. CR1 ist reserviert und darf nicht benutzt werden. CR0 enthält das Maschinenstatuswort und einige zusätzliche Bits, die den Betriebsmodus des Prozessors steuern. Viele davon sind reserviert. Wurde für eine solche zum Beispiel die Paging-Einheit Hardware-mäßig aktiviert (bit 31 = 1), enthält CR2 in dem Fall, daß sich die Page nicht im physikalischen Speicher befindet, die lineare Adresse der Speicherzelle, auf die eigentlich zugegriffen werden sollte. In CR3 befindet sich die Basisadresse des Inhaltsverzeichnisses, welche die Adressen der Page-Tabellen enthält, die sich im Adreßraum einer Task befinden. Bei jedem Task-Wechsel wird automatisch die neue Adresse des Page-Directory geladen.

Die Systemadreß-Register zeigen auf die Anfangsadresse der globalen (GDTR)

und der Interrupt-Deskriptortabelle (IDTR), während das TR-Register den 16-bit-Selektor auf den Deskriptor des augenblicklich aktiven Task-State-Segments und das LDTR denjenigen enthält, der den Deskriptor des Segmentes beschreibt, welches man als lokale Deskriptortabelle bezeichnet.

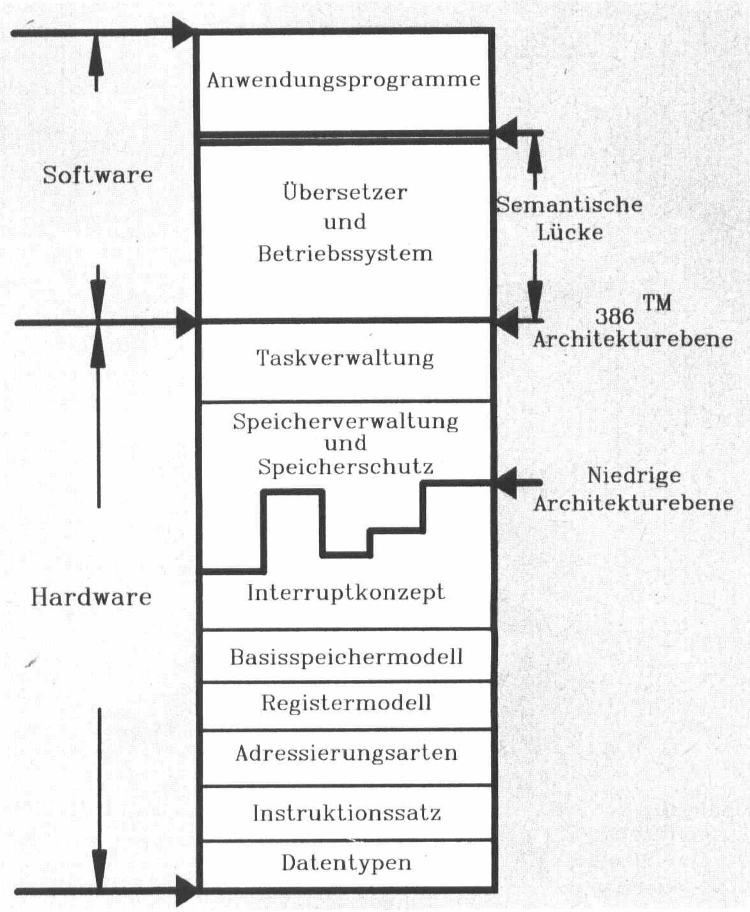
Die Debug- und Testregister

Die Debug-Register können mit dem Inhalt von Hardware-mäßigen Breakpoint-Adressen sowohl von Instruktionen als auch von Daten geladen werden. Ist eine solche Adresse erreicht, erzeugt die Maschine eine Ausnahmebedingung, an deren Ausgang ein Debugger steht. Dieser Debugger sollte seinerseits in der Lage sein, auf die Adresse der Instruktion zu zeigen, welche die Break-Bedingung erfüllte und sie natürlich auch zu disassemblieren. Es ist möglich, Breakpoints auch im EPROM- oder im ROM-Bereich eines Systems zu setzen. Denn es muß nur die Adresse der Instruktion geladen werden, um die Programmausführung zu unterbrechen – die Instruktion für eine solche Unterbrechung braucht nicht durch einen Software-Interrupt oder einen Haltebefehl überschrieben zu werden. Da die Debug-Register auch mit Adressen von Daten geladen werden können, läßt sich auch eine Break-Bedingung für schreibenden oder lesenden, systemweiten oder lokalen Zugriff auf Daten verschiedener Breite definieren. Auf diese Weise ist man in der Lage, Fehler im Programm zu entdecken, die von sich dynamisch ändernden Systembedingungen abhängig sind; solch ein Fehler ist etwa das Überschreiben in einer Tabelle aufgrund eines falsch berechneten Indexes.

Die Testregister dienen zur Überprüfung der Funktionsfähigkeit des sogenannten Translation-Lookaside-Buffers (TLB), der für x-Pages (implementationsabhängig) die Adreßübersetzungs-Information enthält, die man benötigt, um von einer linearen auf eine physikalische Adresse zu kommen. Beim i486-Prozessor dienen sie darüber hinaus zum Test des On-Chip-Cache.

Das Basis-Speichermodell

Ein Programm zusammen mit einfacher System-Software besteht aus einer Abfolge von Anweisungen, die bei „Start" beginnen und bei „End" aufhören. Es werden in Abhängigkeit von irgendwelchen Daten, die über Eingabegeräte gelesen werden, sequentiell bestimmte Algorithmen gerechnet und die Ergebnisse ausgegeben. Diese simple Umgebung müßte auf ein ebenso einfaches Speichermodell abgebildet werden können, um die Aufgabe möglichst bequem formulieren zu können. Die Programmumgebung erfor-

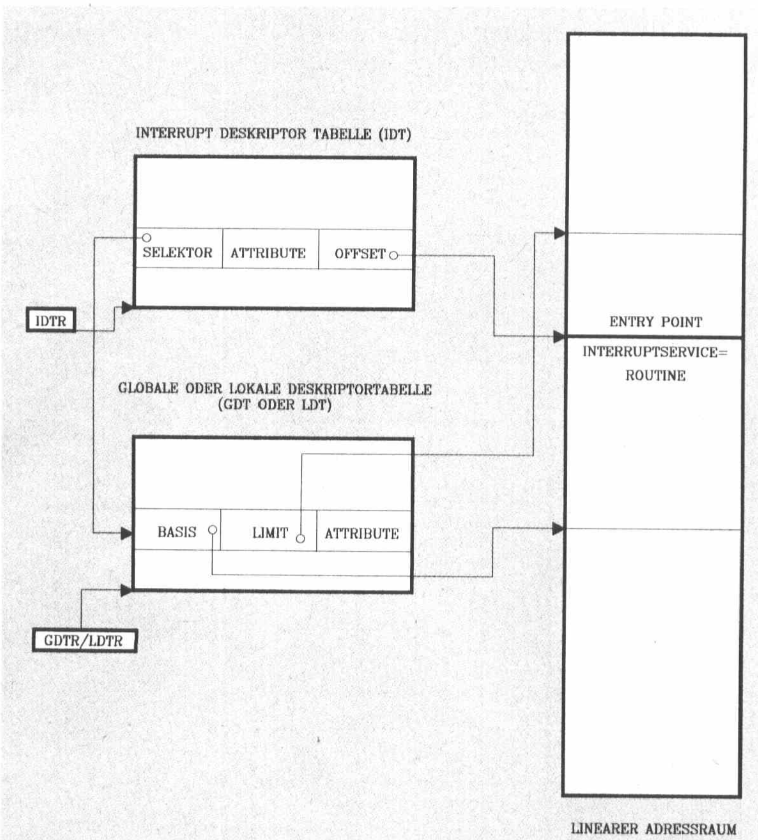


Schematische Darstellung der 386-Architektur

dert nur einen flachen und linearen physikalischen Adreßraum. Die 32-bit-386-Architektur unterstützt einen solchen Adreßraum bis zur Länge von 4 Gigabyte. Die Ein- und Ausgabeperipherie kann über separate Ports angesprochen werden. Insgesamt stellt die 386-Architektur 64 KByte Port-Adressen zur Verfügung. Natürlich ist auch eine Abbildung des Eingabe- und Ausgabe-Adreßraums auf den normalen Speicher (sogenanntes Memory Mapped I/O) möglich.

Ein solches Speichermodell besitzt keinerlei interne Struktur. Programmcode, Daten und Stack befinden sich komplett im linearen Adreßraum, der dem physikalischen Adreßraum entspricht. Eine strukturelle Einleitung des Adreßraumes wird erst in komplexen Anwendungen nötig. Grundsätzlich benutzt die 386-Architektur

eine Adressierungstechnik, die man als Segmentierung bezeichnet. Dabei wird der Speicherbereich in ein oder mehrere lineare Bereiche unterteilt, die man Segmente nennt. Ein Segment kann in den



Übergabe der Programmkontrolle durch ein Interrupt- oder Trap-Gate

Grenzen von 1 Byte bis 4 Gigabyte beliebig lang sein. Damit besteht jede Speicheradresse aus zwei Teilen: aus der Segment-Basisadresse des Segmentes, das die Speicherzelle enthält (es kann ja sehr viele geben), und einem Relativteil (Offset), der innerhalb des Segmentes auf die adressierte Speicherzelle zeigt. Das einfachste Speichermodell erhält man, wenn jedes Segmentregister (CS, SS, DS, ES, FS und GS) auf einen Deskriptor zeigt, dessen Basisadresse null ist. Beträgt die Segmentgrenze für alle identisch 4 Gigabyte, erhält man das einfache lineare physikalische Speichermodell. Die Segmentierung wurde sozusagen logisch abgeschaltet, während Paging über ein Bit im CR0-Register gesteuert wird. Die logische Adresse entspricht in diesem einfachen Modell der physikalischen.

Interrupt-Konzept und Ausnahmebedingungen

Jedes Rechnersystem muß in der Lage sein, auf Unterbrechungen zu reagieren, die durch das asynchrone, nicht vorhersagbare Auftreten eines Ereignisses ausgelöst werden. Ferner muß die Rechnerarchitektur ein Konzept bereitstellen, das definiert, was bei Ausnahmebedingungen zu geschehen hat, die systemweite Konsequenzen nach sich ziehen. Erstere nennt man „Interrupts" und letztere „Exceptions". Beispiele für Interrupts sind das Betätigen eines Tastaturzeichens oder Meldungen eines Eingabe-/Ausgabegerätes, das die Daten bereitgestellt hat, die das System benötigt. Ohne eine Instruktion auszuführen, wird der Programmablauf vollständig geändert. Ausnahmebedingungen werden als Folge einer Befehlsausführung erzeugt, nämlich dann, wenn bestimmte Bedingungen die ordnungsgemäße Ausführung nicht zulassen – die Maschine stößt beispielsweise auf einen Operationscode, der nicht interpretierbar ist, also nicht zum Befehlsvorrat gehört, oder es wurde eine Page adressiert, die sich nicht im physikalischen Speicher befindet. Man unterscheidet drei Klassen von Exceptions.

Ein „Fault" ist eine Ausnahmebedingung, die vor oder während der Ausführung einer Instruktion gemeldet wird. CS und EIP und Quelloperanden werden gerettet und zeigen auf den Befehl, der die Fault-Bedingung verursachte, bevor die Kontrolle der Service-Routine übergeben wird. Alle diese Instruktionen können unter den exakt gleichen Bedingungen erneut gestartet werden.

Eine „Trap-Bedingung" wird nach der Ausführung eines Befehls gemeldet. CS und EIP zeigen in diesem Fall auf die Instruktion, die dem Befehl folgt, der das Trap erzeugt hat. Die verursachende Instruktion wird ausgeführt, so daß sich Register oder Speicherinhalte verändert haben können.

Eine „Abort-Exception" wird unter schwerwiegenden Systemfehlern generiert. Fehlerhafte Systemtabellen können der Grund dafür sein. Die Programmausführung kann unter diesen Umständen nicht wieder aufgenommen werden. Eine Service-Routine für diese Fälle wird wahrscheinlich das Betriebssystem neu starten.

Die 386-Architektur verwaltet sowohl Interrupts als auch Exceptions in gleicher Weise. Jede Interrupt-Quelle und jede Ausnahmebedingung trägt eine Nummer von 0 bis 255, um sie voneinander unterscheiden zu können. Da man nicht wissen kann, wie viele Interrupt-Quellen sich in einem System befinden und welche Wich-

tigkeit sie haben, legt der System-Designer die Zuordnung von Nummer und Interrupt-Typ fest, nicht aber die Architektur. Andererseits werden die Ausnahmebedingungen in der 386-Architektur definiert. Sie belegen bestimmte Werte in einer Tabelle (0 bis 31), die für beide Unterbrechungsarten gleichermaßen gilt.

Grundsätzlich sind die Einträge 0 bis 31 für Exceptions reserviert; alle anderen kann der System-Designer frei verwenden. Die Nummer selbst stellt einen Index auf den Eintrag in die Tabelle dar. Sie wird Interrupt-Deskriptor-Tabelle (IDT) genannt und ist systemweit nur einmal vorhanden; sie kann sich irgendwo im Speicher befinden. Zur Urladezeit wird ihre Adresse in das IDRT-Register mittels der LIDT-Instruktion geladen. Die IDT enthält eine Anzahl von Deskriptoren (maximal 256).

Der Typ Gate

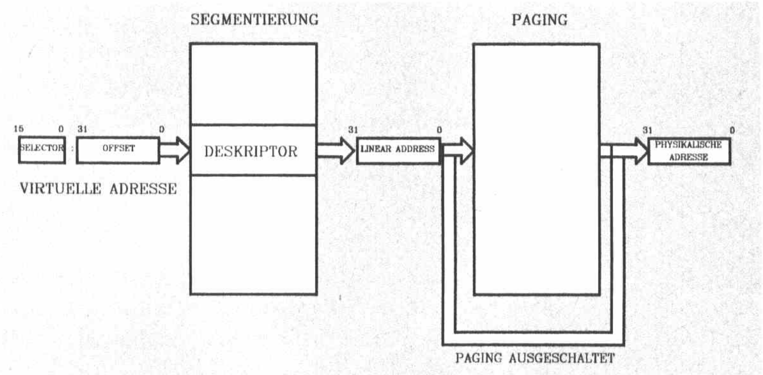
Diese Deskriptoren müssen vom Typ Gate sein (der Begriff Gate wird später im Zusammenhang mit der Vertrauenshierarchie der 386-Architektur näher erläutert). Das Gate seinerseits kann entweder vom Typ Trap, Interrupt oder Task sein. Der Systemprogrammierer muß dafür sorgen, daß an den ensprechenden Einträgen der richtige Gate-Deskriptor steht, dessen Selektoranteil seinerseits auf einen Deskriptor in der GDT oder LDT zeigt, wo die Basisadresse des Codesegmentes steht, in dem sich die Service-Routine befindet. Der Offset-Anteil des Gate-Deskriptors in der IDT wird zu dieser addiert, um die genaue Einsprungadresse in die Service-Routine zu ermitteln, die das Interrupt oder die Ausnahmebedingung behandelt. Unsere Abbildung (Seite 10) zeigt diesen Zusammenhang für Trap- und für Interrupt-Gates schematisch.

Der einzige Unterschied zwischen Trap- und Interrupt-Gates ist das Setzen des IF-Bits (Interrupt-Flag) im EFLAGS-Register. Geht man durch ein Interrupt-Gate, wird IF unabhängig von seinem augenblicklichen Wert auf Null gesetzt, das heißt, damit wird auf weitere Unterbrechungsanforderungen nicht mehr reagiert. Geht man durch ein Trap-Gate, wird das IF-Bit nicht verändert. Gleiches gilt für Task-Gates. Die 386-Architektur erlaubt für diese Fälle weitere (geschachtelte) Unterbrechungen. Die IRET-Instruktion führt zur Wiederaufnahme des unterbrochenen Programms.

Speicherverwaltung und -schutz

Auch in Ein-Benutzer-Umgebungen möchte der Anwender heute möglichst mehrere Aufgaben gleichzeitig erfüllen. Ein serielles Laden der einzelnen Programme mit anschließendem Warten auf das jeweilige Ergebnis ist jedoch mühsam.

Eine formal beschriebene Aufgabe kann recht gut mit einer sequentiellen geschlossenen Einheit von Programmcode, einer Task beschrieben werden. Eine Task besitzt ihren eigenen logischen Adreßraum. Ist sie völlig unabhängig von anderen, soll sie keinerlei Betriebsmittel wie Code, Daten, Stack usw. mit anderen Tasks teilen. Ist sie es nicht, geschieht ein Teilen der Betriebsmittel nur unter „Aufsicht" des Systemprogrammierers, der das Betriebssystem entworfen hat.



Übersetzungsmechanismus der 386-Architektur von einer virtuellen in eine physikalische Adresse

Adreßbildung

Der physikalische Speicher der 386-Architektur besteht aus 2^{32} hintereinanderliegenden Byte. Jedes dieser Byte besitzt eine distinkte Adresse, die man als physikalische Adresse bezeichnet. Sie muß keineswegs der Adresse entsprechen, die der Programmierer sieht. Sie besteht, wie bereits genannt, aus zwei Teilen: 1. aus einer Segmentadresse, die das Segment beschreibt, in der das Byte liegt, auf das zugegriffen werden soll; 2. aus einem relativen Offset-Anteil, der genau auf die Speicherstelle zeigt, die etwa gelesen werden soll. Diese Adresse wird virtuelle Adresse genannt. Um nun zur physikalischen Adresse zu gelangen, durchläuft sie einen Übersetzungsmechanismus, der für die Abbildung der virtuellen auf die physikalische Adresse zuständig ist.

Dieser Mechanismus kann nicht nur die Abbildungsfunktion wahrnehmen, sondern darüber hinaus auch prüfen, ob die virtuelle Adresse überhaupt in eine physikalische transformiert werden darf

oder ob der Zugriff auf einen speziellen physikalischen Speicherbereich in der Art und Weise, wie er gefordert wurde, überhaupt gestattet ist. Ist das nicht der Fall, erfolgte also die Adressierung ohne die dafür notwendige Berechtigung, dann definiert die 386-Architektur das Erzeugen einer sogenannten Ausnahmebedingung (Exception 13 = „General Protection Fault").

Mechanismen

Die Mechanismen, die in der 386-Architektur festgelegt wurden, heißen Segmentierung und Paging. Beide Techniken benutzen zur Adreßübersetzung Tabellen, die im Speicher abgelegt wurden und auf die nur das Betriebssystem Zugriff hat. Einem Anwendungsprogramm ist es nicht erlaubt, diese Tabellen zu manipulieren. Typischerweise wird das Betriebssystem für jede Task einen anderen Satz von Übersetzungstabellen führen; dadurch steht jeder Task der gesamte virtuelle Adreßraum zur Verfügung. Darüber hinaus besitzt jede Task ihren eigenen virtuellen Adreßbereich. Die Tasks können einander also nicht ins Gehege kommen. Die Abbildung zeigt, wie der zweiteilige virtuelle Adreßraum mittels Segmentierung in den linearen Adreßraum übersetzt wird. Wurde der Paging-Mechanismus abgeschaltet (Bit 31 im CR0-Register), entspricht der lineare Adreßraum dem physikalischen, da die Abbildung eins zu eins vollzogen wurde – wenn nicht, dienen Page-Tabellen zur weiteren Übersetzung.

Während der virtuelle Adreßraum zweidimensional ist, besitzt der lineare nur eine Dimension; er weist dieselbe Struktur auf wie der physikalische. Beide sind 4 Gigabyte groß. Im Gegensatz dazu enthält der virtuelle Adreßraum bis zu 16K-Segmente, von denen jedes einzelne 4 Gigabyte lang sein kann. Also beträgt sein Umfang insgesamt 64 Terabyte. Segmentierung und Paging sind zwei voneinander völlig unabhängige Abbildungsmechanismen. Beide können auch Zugriffsrechte untersuchen. Warum begnügt man sich nicht mit einem Konzept? Paging ist die beste Methode, um virtuelle Speichersysteme zu verwalten, in denen dem Benutzer ein riesiges Speichersystem unabhängig vom physikalischen Speicherausbau vorgegaukelt wird. Möchte man dagegen ein komplexes System logisch in variablen Blöcken strukturieren, eignet sich Segmentierung besser.

Schutzmechanismen

Die 386-Architektur unterstützt zwei Klassen von Schutzmechanismen. Zum einen schützt man die Segmente des Betriebssystems vor denen einer Task, zum anderen lassen sich verschiedene Tasks voneinander vollständig isolieren. Der virtuelle Adreßraum der einen kann mittels ihrer Abbildungstabellen einen anderen physikalischen Speicherbereich überdekken als der einer anderen, die über ihre eigenen unabhängigen Abbildungstabellen verfügt. Dabei können sie durchaus identische virtuelle Adressen besitzen. Ihre unterschiedlichen Abbildungstabellen sorgen für unterschiedliche physikalische Adressen. Auf diese Weise ließe sich auch das Betriebssystem in eine eigene Task einbetten und es vor dem Rest der Programmwelt isolieren.

Das in der 386-Architektur festgelegte Konzept ist aber wesentlich wirkungsvoller. Da alle Tasks in einem System möglicherweise auf Betriebssystem-Ressourcen zugreifen müssen, wird diesen Ressourcen ein Teil des virtuellen Adreßbereiches zugänglich gemacht, indem sie die entsprechende Abbildungstabelle miteinander teilen. Diesen Teil des virtuellen Adreßraums, in dem das Betriebssystem gespeichert ist und der von allen Tasks gemeinsam geteilt wird, nennt man globalen Adreßbereich. Im Gegensatz dazu nennt man den virtuellen Adreßraum, der nur zu einer Task gehört und der nicht mit einer anderen geteilt wird, den lokalen Adreßraum. Code und Daten sind Bestandteil der „Privatsphäre" einer Task. Diese Einteilung erlaubt dem Betriebssystem, dem von einer Task beanspruchten Speicher gleiche virtuelle Adressen zu geben, die im physikalischen Adreßraum an voneinander isolierten Bereichen liegen. Andererseits wird eine virtuelle Adresse im globalen Adreßraum von allen im System befindlichen Tasks auf dieselbe physikalische Speicherzelle abgebildet. Dadurch werden sie alle in die Lage versetzt, auf Betriebssystemmittel zuzugreifen.

Das Schutzkonzept einer Task

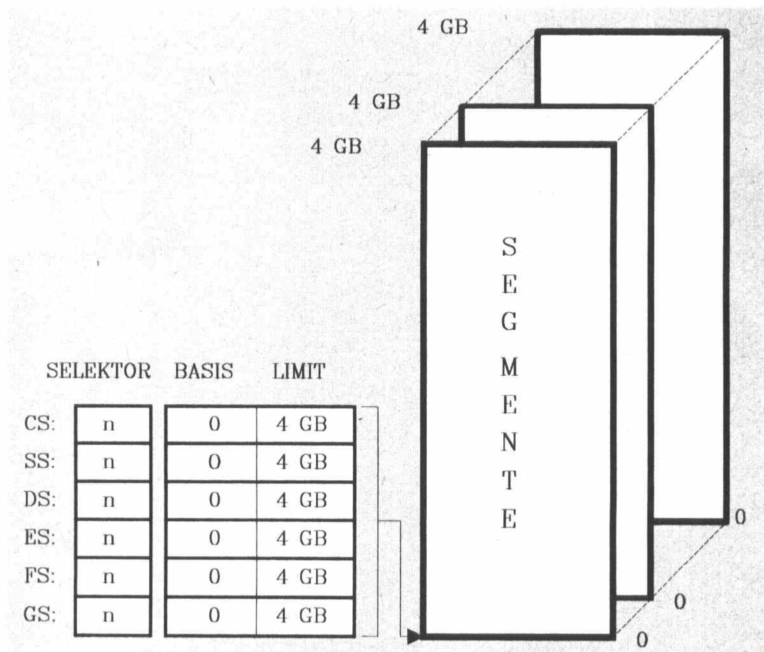
Die 386-Architektur definiert eine vierstufige Vertrauenshierarchie, und zwar in Abhängigkeit von der Wichtigkeit der Programmteile. Man geht davon aus, daß man den Anwenderprogrammen am wenigsten zutraut, fehlerfrei zu sein. Man geht im Gegenteil davon aus, daß sie alles in ihrer Macht Stehende versuchen wer-

den, um das Betriebssystem zu zerstören. Diesen Programmen wird die äußerste Privilegstufe 3 zugewiesen.

Im Gegensatz dazu unterstellt man dem Betriebssystem-Designer ausschließlich besten Willen, seinen Code so betriebssicher wie nur möglich zu gestalten. Da er zudem die Rechnerressourcen für das gesamte System zur Verfügung stellt, billigt man ihm die höchste Wichtigkeit zu und überläßt ihm die innerste Ebene 0 der Vertrauenshierarchie. Der relevanteste Teil des Betriebssystems ist der Kern. Er erhält Privilegstufe 0. Der Rest des Betriebssystems wird auf Stufe 1 verfrachtet. Stufe 2 kann von Schnittstellen-Software oder ähnlichem eingenommen werden. Stufe 3 ist vom Anwender besetzt.

Der Rechner holt und exekutiert die Instruktionen aus dem Segment, das durch den Inhalt des CS-Registers, den Segmentselektor, gekennzeichnet ist. Dem gerade aktiven Segment wird ein CPL (= Current Privilege Level) zugeordnet. Damit ist die Vertrauensebene des Programms festgelegt, das augenblicklich bearbeitet wird. Der CPL definiert, auf welche anderen Segmente im System zugegriffen werden darf und auf welche nicht. Immer dann, wenn dieses Programm etwa ein anderes Datensegment adressiert, wird der CPL mit der Privilegstufe des Segments verglichen, das adressiert wurde. Nur wenn die Privilegstufe des Datensegments, auf das zugegriffen werden soll, numerisch größer oder gleich dem CPL ist, wird der Zugriff auch erlaubt. Ist das nicht der Fall, wird Exception 13 (General Protection Fault) erzeugt, und das Betriebssystem ist gefordert, die illegale Aktion zu behandeln.

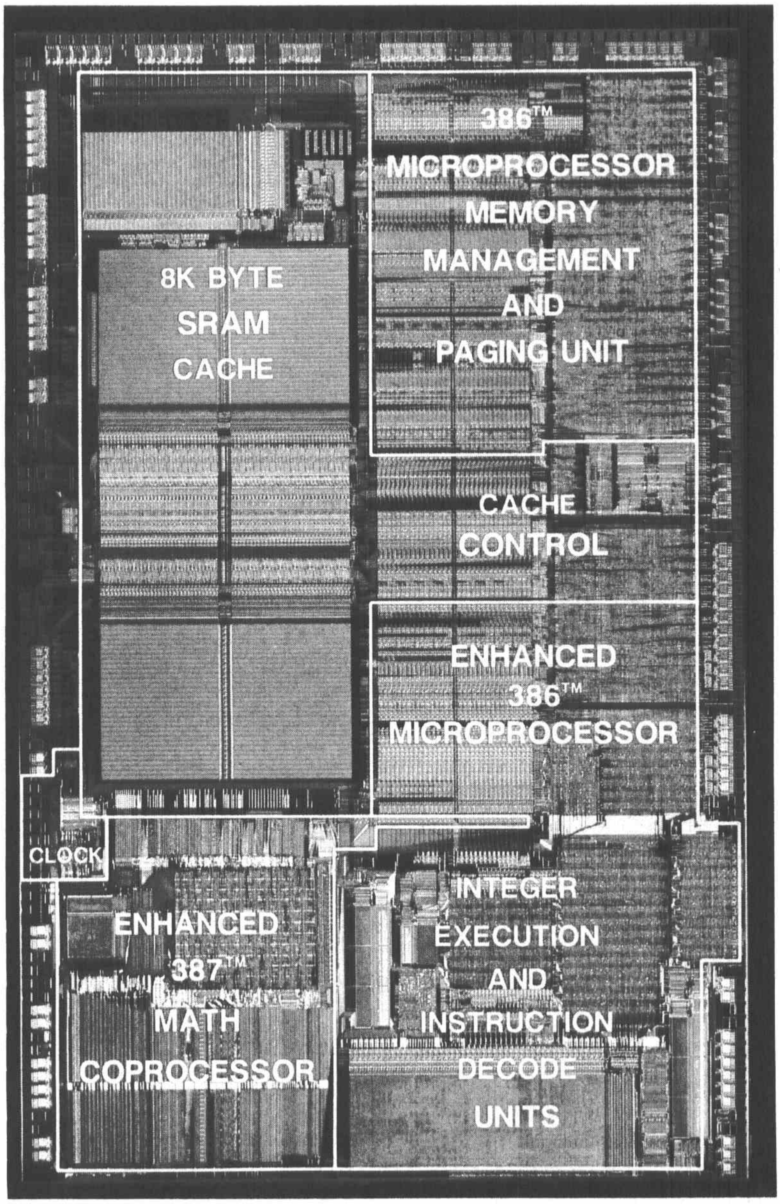
Jede Privilegstufe besitzt ihren eigenen Stack. Augenscheinlich wird das Problem, wie denn von einem Anwenderprogramm, das ganz außen auf Privilegstufe 3 angesiedelt ist, auf den Betriebssystemcode, der sich auf der innersten Privilegstufe 1 befindet, etwa mit einer „Open-File"-Anforderung zugegriffen werden kann. Die oben genannte Regel verbietet eigentlich einen solchen Zugriff. Um als Anwender trotzdem an die Betriebssystem-Ressourcen heranzukommen, stellt die 386-Architektur den Mechanismus der „Call Gates" zur Verfügung. Call Gates sind Deskriptoren, die ähnlich wie bei der Interrupt-Verarbeitung den kontrollierten Zugang zu inneren, das heißt numerisch kleineren Privilegstufen zulassen. Da der Betriebssystementwickler alle diese Gates im System festlegt, kann man sicher sein, daß sie die Eintrittspunkte für einige wenige Prozeduren definieren, die die Tasks benötigen, beispielsweise den Zugriff auf das Eingabe-/Ausgabesystem oder die Allokierung von Speicher. Die Programmflußänderung durch ein Call Gate ist mit einem indirekten Prozeduraufruf vergleichbar. Für den ordnungsgemäßen Übergang von einer äußeren Privilegstufe in eine innere gilt die Regel, daß die Privilegstufe des Deskriptors für das Call Gate (DPL = Deskriptor Privilege Level) numerisch größer oder gleich dem Maximum aus der augenblicklichen Privilegstufe des Segments der aufrufenden Task (CPL = Current Privilege Level) und der Privilegstufe des Selektors ist, der das Call Gate spezifiziert (RPL = Requestor's Privilege Level). Call Gates, die dem Anwender Betriebssystemmittel zur Verfügung stellen wollen, sollten also den DPL = 3 besitzen.



Flaches, lineares Speichermodell

Bleibt noch zu klären, wie denn der Betriebssystemprozedur Parameter durch die rufende Task zur Verfügung gestellt werden können, der Übergang durch das Call Gate erfolgt ja frei von Parametern. Wenn ein „Open File" durchgeführt werden soll, muß das Betriebssystem wenigstens wissen, welche Datei gemeint ist. Jede Privilegstufe besitzt ihren eigenen Stack, um auch hier möglichst hohen Systemschutz zu gewährleisten. Der Deskriptor eines Call Gates spezifiziert nun

einen „Count"-Parameter. Es werden beim Übergang durch ein Call Gate genau „Count"-32-bit-Doppelworte vom alten Stack in das neue kopiert, so daß die gerufene Prozedur die benötigten Parameter auf ihrem Stack vorfindet.



Blockschaltbild des 80386-Prozessors

Segmentierung

Die Segmentierung kann den virtuellen Adreßraum der 386-Architektur in eine Menge logischer Bereiche verschiedener Länge organisieren. Eine virtuelle Adresse besitzt zwei Teile: einen Teil, der das Segment identifiziert, und einen Teil, der die relative Adresse innerhalb eines Segments spezifiziert. Segmente sind die Grundbausteine, auf die eine Abbildungsfunktion wirkt, um von einer virtuellen zu einer linearen Adresse zu kommen. Jedes Segment im System wird durch drei Parameterfelder beschrieben:

1. Die Basisadresse gibt an, wo das Segment im linearen Adreßraum anfängt. Sie entspricht der virtuellen Adresse mit einem Offset von 0.

2. Die Segmentgrenze kennzeichnet die maximale Länge eines Segments. Der Offset-Anteil der virtuellen Adresse darf die Segmentgrenze nicht überschreiten.

3. Das Attributfeld charakterisiert die Eigenschaften eines Segments. Hier werden die Zugriffsmöglichkeiten eines Programms auf das Segment festgelegt; beispielsweise darf das Segment beschrieben oder nur gelesen oder nur ausgeführt werden. Auch die Privilegstufe, die dem Segment zugeordnet wird, ist im Attributfeld verschlüsselt.

Die ein Segment beschreibenden Charakteristika sind in einem Deskriptorfeld zusammengefaßt. Deskriptoren werden in Tabellen abgelegt und dort verwaltet. Sie können dynamisch erzeugt und gelöscht werden. Die Deskriptoren, die systemweit zur Verfügung stehen sollen, befinden sich in der GDT (Global Descriptor Table); solche, die zum lokalen logischen Kontext einer Task gehören, sind in der LDT (Local Descriptor Table) abgelegt. Der virtuelle Adreßraum ist also in Hälften unterteilt. Die Abbildungsfunktion für die eine Hälfte wird von der GDT, die für die andere von der LDT übernommen. Der virtuelle Adreßraum besteht maximal aus 2^{14} Segmenten, entsprechend der Anzahl der möglichen 2^{13} Einträge in jeweils die GDT und LDT. Jedes Segment kann 2^{32} Byte lang sein. Damit berechnet sich der maximale virtuelle Adreßbereich zu 2^{46} Byte entsprechend 64 Terabyte.

Welches der vielen Segmente adressiert werden soll, entscheidet der erste Teil der virtuellen Adresse, nämlich der Selektor. Er stellt einen Index in die GDT oder LDT hinein dar. Man benötigt 13 bit, um den Index in die GDT oder LDT hinein darzustellen, und zwar die Bits 3 bis 15; Bit 2 entscheidet, für welche der beiden Tabellen. Ist es 1, adressiert der Index die LDT; ist es 0, wird ein Deskriptor in der GDT gesucht. Die Bits 0 und 1 verschlüsseln den RPL (Requestor's Privilege Level). Die 386-Architektur definiert, daß – sobald ein Zugriff auf ein Segment über den Selektor unternommen wird – ein Vergleich des CPL (Current Privilege Level) des aktiven Programms mit dem DPL des Segmentes durchgeführt werden

muß, auf das zugegriffen werden soll. Das RPL-Feld im Selektor modifiziert diesen Test, indem das Maximum aus CPL und RPL gebildet und dieser Wert dem augenblicklich aktiven Programm zugeordnet wird. Der DPL des Segmentdeskriptors muß numerisch größer oder gleich dem MAX(CPL,RPL) sein, damit der Zugriff erlaubt wird. Der CPL entspricht normalerweise dem DPL (Descriptor Privilege Level) des Segments, in dem sich das augenblicklich bearbeitete Programm befindet. Der CPL ändert sich automatisch, wenn die Kontrolle an ein Segment mit unterschiedlichem DPL übergeben wird. Ein Zugriff ist nur dann erlaubt, wenn der DPL des Zielsegments numerisch größer oder gleich dem Maximum aus CPL und RPL des Selektors ist.

Paging

Als zweiten Mechanismus der Speicherverwaltung legt die 386-Architektur das Paging-Konzept fest, um die virtuelle Adresse endgültig in eine physikalische Adresse zu übersetzen. Während die Segmentierung aus einer virtuellen Adresse eine lineare macht, transformiert Paging die lineare in eine physikalische. Die Segmentierung arbeitet mit Segmenten variabler Länge, Paging mit Bereichen, die eine fixe Länge besitzen. Diese werden Pages genannt. Irgendeine Page im linearen Adreßraum kann auf eine beliebige Page identischer Länge im physikalischen Speicher an einer unterschiedlichen Adresse abgebildet werden.

Pages in den derzeitigen Implementationen der 386-Architektur sind 4 Kilobyte lang. Das Paging-Konzept unterteilt also den 2^{32} Byte großen linearen Adreßraum in 2^{20} Pages. Da Pages immer auf 4-Kilobyte-Adreßgrenzen liegen, entsprechen die unteren 12 bit der physikalischen Adresse. Sie werden sozusagen durchgereicht. Die Abbildungsfunktion wird über die oberen 20 bit vollzogen. Den höherwertigen 20 bit der linearen Adresse können bestimmte Attribute zugewiesen werden, die eine Übersetzung in eine physikalische Adresse nicht erlauben, wenn das Betriebssystem dies so festgelegt hat. Sie befinden sich in den Bit 0 bis 11 eines Pagetable-Entry's (PTE). Wird trotzdem ein Zugriff versucht, muß das Betriebssystem das Programm terminieren. Wenn sich andererseits aber eine adressierte Page nicht im Speicher des Systems befindet, sondern auf eine Platte ausgelagert wurde, wird eine Ausnahmebedingung erzeugt. Das Betriebssystem muß dafür Sorge tragen, daß die benötigte Page in den Speicher kopiert wird, damit das Programm auf sie zugreifen kann.

Die Abbildungsfunktion der linearen in eine physikalische Adresse wird mittels Übersetzungstabellen vollzogen. Die 386-Architektur unterstützt eine Struktur in zwei Ebenen. Die Einträge im Page-Inhaltsverzeichnis zeigt auf die Anfangsadressen der Page-Tabellen. Die Einträge in der Page-Tabelle zeigen auf die Anfangsadresse der Page im physikalischen Speicher, zu der die unteren 12 bit addiert werden, um zu der adressierten Speicherzelle innerhalb einer Page zu gelangen. Im Gegensatz zur Segmenttabellen-Struktur lassen sich die Page-Tabellen nicht so einfach in einen globalen und in einen lokalen Bereich teilen. Trotzdem ist es möglich, daß alle im System befindlichen lokalen Tasks einen globalen Bereich des linearen Adreßraums miteinander teilen, indem sie dieselbe Adreßübersetzungs-Information benutzen.

Task-Verwaltung

Die 386-Architektur ist so entworfen worden, daß nicht nur Einzeltask-Systeme, sondern auch mehrere Tasks im System von Anfang an unterstützt werden. Die Architektur „weiß", was man unter einer Task zu verstehen hat. Zu diesem Zweck werden Task-relevante Systemstrukturen festgelegt. Dabei handelt es sich um das

- Task-State-Segment (TSS),
- den Task-State-Segment-Deskriptor,
- das Task-Register (TR) und
- den Task-Gate-Deskriptor.

Mit Hilfe der in diesen Systemstrukturen enthaltenen Information ist es sehr einfach, von einer Task in eine andere zu wechseln. Die jeweilige Implementation der 386-Architektur vollzieht diesen Wechsel in Hardware. Der Systemprogrammierer sorgt durch Verwendung eines geeigneten Befehls nur für den Übergang von der alten zur neuen Task. Die notwendige Detailarbeit übernimmt die Hardware der Maschine. Bei jeder Task-Wechsel-Operation wird der gesamte Zustand der Maschine gerettet, das heißt alle Register, der Adreßraum und ein Verbindungszeiger zur vorherigen Task. Es wird der Ausführungszustand der neuen Task geladen, Zugriffsrechte werden geprüft, und mit der Bearbeitung der neuen Task wird begonnen.

Literaturliste

J.H. Crawford und P.P. Gelsinger, Programming the 80386, Sybex 1986, 1987

Intel, 386-Microprocessor, Datasheet, Nov. 1988, Order Code 231630

Intel, 80386 System Software Writer's Guide, 1987, Order Code 231499

Intel, 80386 Programmer's Reference Manual, 1987, Order Code 230985

Intel, i486-Microprocessor, Datasheet, April 1989, Order Code 24044

H. Dau, Private Kommunikation

J. Bodenkamp, Programmierung des Intel 80386, CHIP Tool, Ausgabe 12, Dezember 1988, Seiten 40 – 45

J. Bodenkamp, Der 80486: Cache und Coprozessor mitintegriert, Elektronik, Ausgabe 8, April 1989

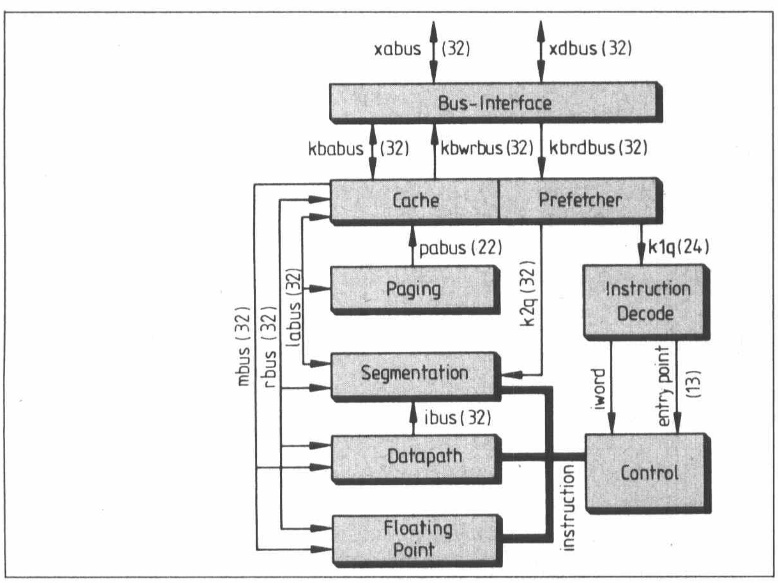
J. Bodenkamp, Der Prozessor 80486: Aufbruch zu neuen Ufern, MC, Mai 1989

Die neue Chip-Generation

Dr. Jens Bodenkamp

Bei den derzeit verfügbaren Implementationen der 386-Architektur handelt es sich um die Prozessoren 80386SX, 80386DX und i486.

Bei der Entwicklung des i486 wurden im wesentlichen drei Hauptziele verfolgt. Erstens sollte die i486-CPU als Implementation der 386-Architektur vollständig binär aufwärtskompatibel zu seinen Vorgängern, dem 80386SX und dem 80386DX, sein. Zweitens wurde diese Architektur so implementiert, daß unter der Randbedingung der Kompatibilität eine möglichst hohe Leistungssteigerung erzielt werden konnte. Die Rechengeschwindigkeit sollte um einen Faktor 3 höher liegen als die des 80386DX bei gleicher Taktfrequenz. Zu diesem Zweck wurde nicht nur die Mikroarchitektur der 386-Maschine neu implementiert, sondern auch die Bus-Schnittstelle verbessert und ein Cache-System auf den Chip geholt. Man gestaltete die kritischen Pfade auf der i486-CPU so, daß höhere Taktfrequenzen und somit bessere Rechenleistungen möglich wurden.



Blockschaltung des 80486-Mikroprozessors

Das dritte Design-Ziel der Ingenieure bestand darin, einige Architekturerweiterungen vorzunehmen, die insbesondere den Aufbau von Multiprozessor-Systemen auf der Basis des i486-Prozessors vereinfachen sollten.

Die Abbildung zeigt ein Blockdiagramm des i486-Prozessors. Man erkennt die bereits beschriebenen Funktionen der 386-Architektur. Die Ausführungseinheit bearbeitet den Instruktionssatz der 386-Architektur, die numerische Architekturerweiterung des 80387 ist auf den Chip gewandert, was zu erheblich mehr Rechenleistung (drei- bis fünfmal schneller als das Gespann 80386DX/80387) für numerisch anspruchsvolle Programme sowie zu Kosten- und Platzersparnis auf der Platine führt. Die Segmentierungseinheit sorgt für die Transformation der virtuellen in eine lineare Adresse, und die Paging-Einheit nimmt sich der Übersetzung der linearen in eine physikalische an. Die beschriebenen Speicherschutz- und Speicherverwaltungs-Konzepte wurden in diesen Funktionsblöcken realisiert, ebenso der Taskwechsel-Mechanismus.

Der Bus des i486-Prozessors

Intel hat versucht, den Bus der i486-CPU dem des 80386DX so ähnlich wie möglich zu machen, um die Erfahrungen der Design-Ingenieure zu bewahren. Trotzdem ist einiges an Funktionalität hinzugekommen. Es nützt die schnellste CPU der Welt nichts, wenn ein langsamer Bus den Zugriff auf Daten, Operanden und Instruktionen bremst. Die Leistung der CPU und die mit ihr assoziierte Bus-Schnittstelle müssen in einem ausgewogenen Verhältnis stehen. Während man die Leistung der CPU beispielsweise in Mips (million instructions per second) mißt, ist für den Bus die Bus-Bandbreite der leistungsrelevante Parameter. Der 80386DX-Prozessor unterstützt eine in zwei Takten synchrone Bus-Schnittstelle. Ein Takt wird minimal benötigt, um die Art des Bus-Zyklus zu definieren und die Adressen der anzusprechenden Speicherzelle auf die entsprechenden Leitungen zu legen. Ein weiterer Takt ist erforderlich, um das Datum in den Prozessor zu lesen oder auf den Speicher zu schreiben. Da der Bus des 80386DX 32 bit breit ist, lassen sich in zwei Takten maximal vier Byte über die Bus-Schnittstelle transportieren. Damit ergibt sich bei einer 25-MHz-Version des Prozes-

sors eine maximale Bus-Bandbreite von 50 Megabyte pro Sekunde.

Behielte man das Bus-Konzept des 80386DX bei, wäre die Bandbreite zu niedrig, um die wesentlich schnellere i486-CPU mit Daten zu füttern. Deshalb unterstützt der Bus des i486-Prozessors die sogenannte Burst-Betriebsart. In diesem Modus wird nur einmal eine Adresse auf die entsprechenden Bus-Leitungen gelegt, und in jedem weiteren Takt werden sukzessive Daten zum Beispiel in das Cache-System auf den Chip gelesen. Es entfallen also die weiteren Adreßtakte. In insgesamt fünf Takten (einer für die Adresse, vier für die Daten) lassen sich 16 Byte über den Bus transportieren (4 Byte pro Takt). Damit ergibt sich die maximale Bus-Bandbreite des i486-Prozessors zu 80 Megabyte pro Sekunde bei einer Arbeitsfrequenz von 25 MHz oder ein Leistungsgewinn von 40 Prozent gegenüber dem 80386DX.

Bus-Details

Diese Bus-Bandbreite wird intern auf dem Chip noch erheblich erhöht (Faktor 5 = 400 Megabyte pro Sekunde). Der Bus zwischen einem internen Cache-System und einem Prefetch-Puffer ist 128 bit breit. Hier können 16 Byte in nur einem Takt in den Puffer geladen werden. Gesteuert wird der Burst-Modus durch die Signale BRDY# (# = active low Signal) und BLAST#. Je weniger der Bus von der CPU benutzt wird, desto mehr Bus-Bandbreite steht anderen Funktionseinheiten, die den Bus mit der CPU teilen müssen, zur Verfügung. Bis auf Interrupt- und Haltzyklen können alle anderen die Burst-Betriebsart benutzen.

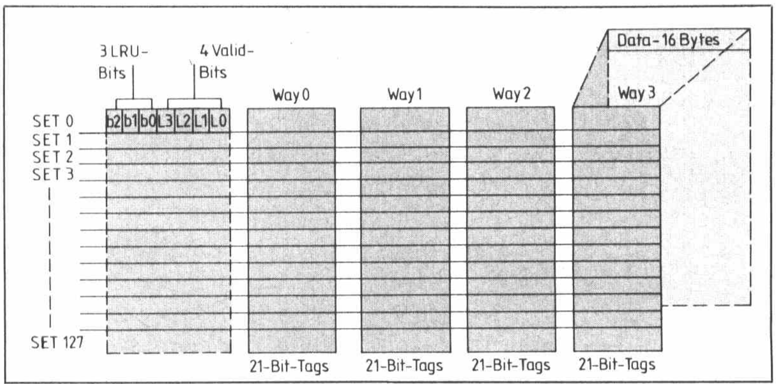
Ein Rechnersystem kann 8 bit breite und 16 bit breite Peripheriegeräte enthalten. Diese Geräte können ihre Daten direkt auf die 32 Datenleitungen des Busses legen. Wo sie sich befinden, wird durch die Signale CS8 und CS16 gesteuert. Zusätzlich zu dem bekannten „HOLD"/„HOLDA"-Protokoll werden mit BREQ und BOFF# Multimaster-Systeme unterstützt. Ein anderer Busmaster im System bittet den Prozessor, durch Anlegen des HOLD-Signals den Bus freizugeben. Dieser bestätigt die Freigabe mit HOLDA. Ein solches Verhalten kann man als eine höfliche Form der Verhandlungsführung zwischen den beiden bezeichnen. Die unhöfliche Art besteht darin, einfach das BOFF#-Signal aufzuschalten. Ein eingeleiteter Bus-Zyklus der i486-CPU wird im nächsten Takt abgebrochen und so lange mit der Wiederaufnahme desselben Zyklus gewartet, bis der andere Busmaster den Bus freigegeben, d.h. das BOFF# zurückgenommen hat. Auf diese Weise lassen sich Deadlock-Situationen in einem Multima-

Tabelle 1. Burst-Ordnung

1. Adresse	2. Adresse	3. Adresse	4. Adresse	
0	4	8	C	1. 64-Bit-Grenze
4	0	C	8	
8	C	0	4	2. 64-Bit-Grenze
C	8	4	0	

Die Reihenfolge der eingelesenen Daten in der Burst-Betriebsart hängt von der ersten Adresse ab

ster-System vermeiden oder auch ECC-Konzepte realisieren. Durch BREQ teilt der i486-Prozessor der Außenwelt mit, daß seine On-Chip-Ressourcen (Cache-System) zum Weiterarbeiten nicht mehr ausreichen und daß er unbedingt den Bus braucht. Mit BREQ kann er die Freigabe erzwingen. Im Gegensatz zu Rechnern, die auf der 80386DX- oder SX-CPU basieren, entspricht der Takt des TTL-kompatiblen Taktgenerators dem Systemtakt. Intern wird diese Uhr geteilt und phasenverschoben, um auf dem Chip genauso viele steigende und fallende Flanken zu haben, um Schaltfunktionen auszulösen, als wäre die äußere Taktfrequenz doppelt so hoch. Ein 25-MHz-i486-Prozessor benötigt also keinen 50-MHz-Taktgenerator, sondern nur einen, der 25 MHz erzeugt. Dadurch wird das System-Design erheblich erleichtert.

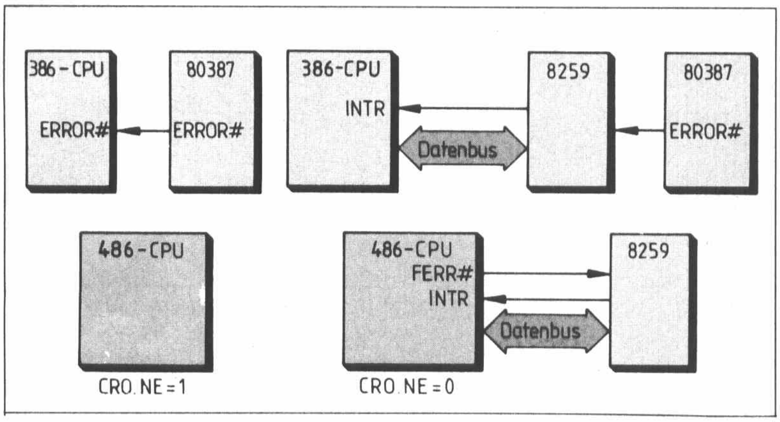


Organisation des 486-Onchip-Cache (8 KByte)

Die Datenleitungen des i486-Prozessors sind in 8-bit-Qualitäten paritätsgeschützt. Die Buskontroll-Logik generiert automatisch die Paritätsbits (Schreibzyklus) und prüft beim Lesen. Liegt eine Verletzung vor, wird dies der Außenwelt über ein Pin (PCHK#) mitgeteilt. Es obliegt dem System-Designer, geeignete Maßnahmen für diese Ausnahmebedingungen zu ergreifen.

Das On-Chip-Cache-System

Das Prinzip der „lokalen Referenz" bestimmt den Wirkungsgrad des Cache. Dieser Grad ist definiert als das Verhältnis zwischen Treffer – eine Kopie der adressierten Speicherzelle wurde im Cache gefunden – und der Anzahl der Zugriffe auf den Cache-Speicher. Je mehr Lokalität ein Programm aufweist, das heißt, je stärker es in mehr oder weniger großen Schleifen organisiert ist, die wiederholt auf Daten und Instruktionen zugreifen, desto höher ist der Wirkungsgrad des Cache. Wenn ein Programm umgekehrt aus einem sequentiellen Strom von Daten und Instruktionen besteht, auf die jeweils nur ein einziges Mal zugegriffen wird, ist die Wirkung des Cache praktisch gleich Null.



Die beiden Arten der Fehlerbehandlung bei Gleitkomma-Instruktionen: links die Standard-Intel-Methode (oben 386/387, unten 486). Ein Fehler wird über die Error-Leitung gemeldet und über die Ausnahme-Bedingung 16 behandelt. In diesem Fall steht das NE-Bit im Control-Register CR0 auf „1". Ist dieses Bit gleich „0", hat der Benutzer die DOS-kompatible Art der Fehlerbehandlung gewählt.

Der i486-Prozessor besitzt ein 8 Kilobyte großes und 4-Wege-assoziatives physikalisches Adreß-Cache auf dem Chip. Würde es sich um ein logisches Cache handeln, das mit virtuellen Adressen arbeitet, wäre es sehr schwierig, ein leistungsfähiges Multimaster-System aufzubauen. Was in einem 386-System mit einem externen Cache-Controller und SRAM für Daten und Instruktionen außerhalb des Chips aufgebaut werden mußte, ist sozusagen auf den Chip gewandert. Für jeweils 4 Byte in einem Satz werden Gültigkeitsbits geführt, die bei der Adressierung geprüft werden. Ein teilweises Füllen eines Satzes ist damit nicht möglich – es ist auch nicht notwendig, da der Bus des i486-Prozessors die Burst-Betriebsart unterstützt.

Die hohe Assoziativität des Cache verhindert Kollisionssituationen, die leicht in sogenannten Direct-mapped Caches entstehen können. Dies soll kurz erläutert werden. Man kann sich den Hauptspeicher so denken, als wäre er in gleich lange logische Bereiche unterteilt. Jeder dieser Adreßbereiche entspricht der Größe des Cache-Speichers. Ein Direct-mapped Cache kann jetzt an der Adresse „n" eine Kopie eines Datums aus dem Hauptspeicherbereich „i" an der Adresse „n" enthalten. Dieser Eintrag im Cache ist direkt auf den logischen Hauptspeicherbereich „i" abgebildet. Unter der Annahme, daß das Programm in einer Schleife wieder auf die Adresse „n", jetzt aber im Hauptspeicherbereich „i + 1", stößt, muß das vorherige Datum durch das neue ersetzt werden. Es liegt das Gegenteil von einem Cache-Treffer oder Hit, nämlich ein Cache-Fehlversuch oder Miss vor. Im nächsten Schleifendurchgang wird wieder auf die Adresse „n" zugegriffen, allerdings wieder im Adreßbereich „i" usw. Man erkennt, daß das „direct-mapped"-organisierte Cache in dieser hypothetischen Schleife ständig Fehlversuche produziert. Man nennt dieses Problem „Thrashing". Ist das Cache mehrfach assoziativ organisiert – hat man also mehrere Wege in das Cache hinein, die alle dieselbe Adresse „n" aus dem Beispiel haben können – tritt diese Kollision nicht auf. Denn die Speicherzelle „n" aus dem Hauptspeicherbereich „i" kann in den ersten Weg und die Adresse „n" aus dem Hauptspeicherbereich „i + 1" kann in den zweiten Weg eingetragen werden. Für 4-Wege-assoziative Caches ist dieser Konfliktfall sehr selten.

Der Wirkungsgrad eines Cache wird zudem durch seine Satzlänge bestimmt. Im Falle der i486-CPU beträgt diese 16 Byte. Programme tendieren dazu, in Schleifen sequentiell abzulaufen. Erzeugt man beim Programmzählerstand „i" einen Cache-Fehlversuch, ist es sehr wahrscheinlich, daß bei „i + 1", „i + 2" usw. ebenfalls ein Fehlversuch produziert wird. Um diesen Fall zu vermeiden, wird das Cache über seine Satzlänge nicht nur mit der Instruktion „i", sondern auch mit den darauffolgenden aktualisiert. Der Burst-Bus des i486-Prozessors unterstützt eine möglichst wirkungsvolle Aktualisierung des Cache in dem oben genannten Sinne. Um das Cache softwaretransparent zu machen – was bedeutet, daß es den zu verarbeitenden System- und Anwendungsprogrammen egal ist, ob es dieses Cache physikalisch gibt oder nicht –, wurde für Schreibzyklen der „Buffered-Write-Through"-Mechanismus gewählt. Immer dann, wenn die CPU einen Schreibzyklus initiiert, aktualisiert sie die entsprechende

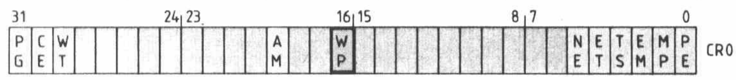
Cache-Adresse und den Hauptspeicher; beide sind seitens der CPU stets auf gleichem Stand. Was passiert aber, wenn ein anderer Busmaster, etwa ein DMA-Controller, neue Daten in den Hauptspeicher schreibt, für die es Kopien im Cache gibt? Auch für diesen Fall wurde Vorsorge getroffen: Werden solche Zyklen im System initiiert, bemerkt das der i486-Prozessor. Er vergleicht die Adressen, auf die geschrieben wird, mit den Adressen in seinem internen Cache und invalidiert diejenigen, die identisch sind. Damit wird beim nächsten Zugriff ein Cache-Fehlversuch erzwungen. Die CPU muß den Hauptspeicher lesen und erhält die aktuellen geänderten Daten.

Zum Schluß der Diskussion des Cache bleibt noch eine Frage offen. Was wird getan, wenn das Cache voll ist – das ist praktisch immer der Fall – und Platz für einen neuen Cache-Satz geschaffen werden muß? Was wird, was darf überschrieben werden? Schlecht wäre es, wenn man etwas überschreibt, das beim nächsten Zugriff benötigt wird. Um diesen Fall zu vermeiden, implementiert der i486-Prozessor einen Algorithmus – er sorgt dafür, daß der Satz überschrieben wird, der in der Vergangenheit am wenigsten benutzt wurde. Man nennt dieses Prinzip „Least Recently Used (LRU)". Da man für ein echtes LRU 6 bit pro Satz benötigen würde, hat man sich für ein Pseudo-LRU entschieden. Mit ihm werden nur 3 bit verwaltet, ohne dabei wesentlich an Leistungsfähigkeit einzubüßen.

Das Cache ist über einen 128 bit breiten Bus mit einer „intelligenten" Prefetch-Einheit verbunden, um Dekodier- und Ausführungseinheit zur sinnvollen Arbeit anhalten zu können.

Die Pipeline

Alle modernen Architekturen unterstützen Programmverarbeitung in einer Pipeline: Wenn die vielen Funktionen, welche die 386-Architektur bereitstellt, sequentiell abgearbeitet werden müßten, käme man zu einer unakzeptablen Rechnerleistung. Unabhängige Mikromaschinen auf dem Chip müssen sehr viele Dinge parallel ausführen, um die geforderte Rechengeschwindigkeit zu erhalten. Die Abbildung der virtuellen Adresse auf die physikalische mit Überprüfung auf gültigen Zugriff, die Berechnung der effektiven Adresse, die Ausführung des Befehls und vieles andere mehr wird in einer Pipeline gleichzeitig bearbeitet, so daß sie die Zeit, die jeder Einzelschritt benötigt, vor dem Anwender verbirgt. „Draußen" erkennt der Benutzer nur, daß sein Programm im Durchschnitt etwa 1,7 CPU-Takte benötigt, um abgearbeitet zu werden.



U/S	R/W	Supervisor Access WP=0	Supervisor Access WP=1	User Access WP=0	User Access WP=1
0	0	R/W/E	R/E	NONE	NONE
0	1	R/W/E	R/W/E	NONE	NONE
1	0	R/W/E	R/W	R/E	R/E
1	1	R/W/E	R/W/E	R/W/E	R/W/E

80386 (Spalte WP=0) – 80486-Erweiterung (Spalte WP=1)

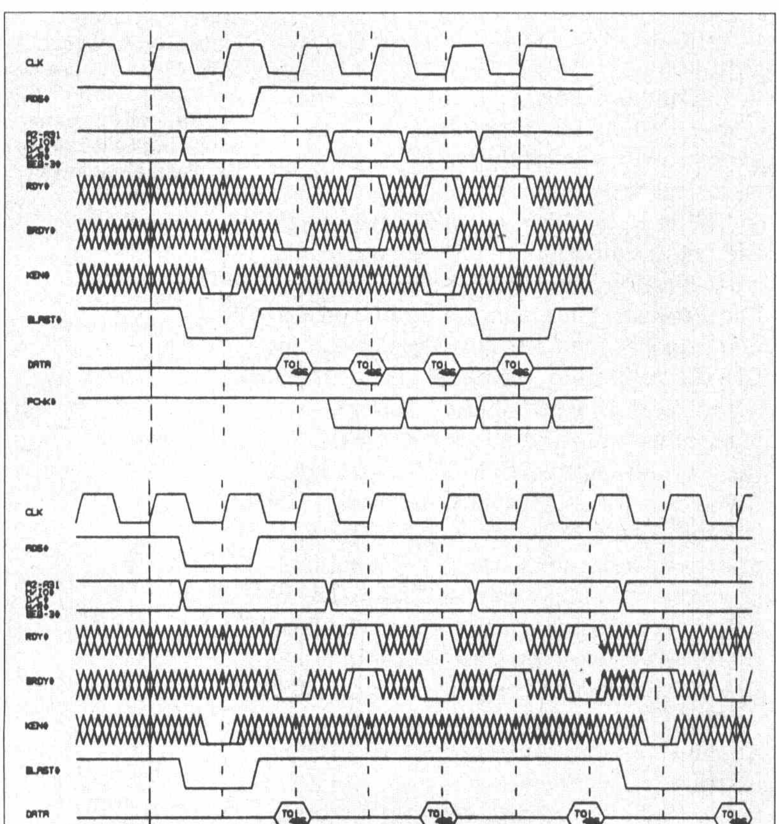
Das WP-Bit (Write Protect) im Control-Register CR0 markiert Pages als schreibgeschützt. Auf diese Weise lassen sich „Copy on Write"- oder „Lazy Copying"-Konzepte im Betriebssystem realisieren.
R = Read;
W = Write;
E = Execute;
U = User;
S = Supervisor

Während die Pipeline der 80386DX-Implementation in vier Stufen organisiert wurde, besitzt der i486-Prozessor eine fünfstufige Pipeline. Sie besteht aus den Schritten

- Instruktions-Fetch
- Instruktionsdekodierung 1
- Instruktionsdekodierung 2
- Instruktionsführung
- „Write Back" (Ergebnisaktualisierung)

Die Integrität der Pipeline stellen Hardware-Interlocks sicher. Für das Herausholen der Instruktion aus dem internen Cache benötigt man nur eine Pipeline-Stufe. Die Dekodierung gestaltet sich etwas komplizierter in zwei Stufen. In der ersten wird durch Interpretation des OP-Codes der Typ der Instruktion identifiziert und in der zweiten die Operandenadresse bestimmt. Beide Hälften werden parallel über zwei separate Busse von einem „intelligenten" Prefetcher gespeist. Beide Dekodierstufen arbeiten ebenfalls überlappend, so daß man auf eine aktuelle Dekodierungsrate von einer Instruktion pro Takt kommt. Bevor die Instruktion zur Ausführung kommt, werden in der Dekodiereinheit zusammen mit dem Kontrollfunktions-Block Maßnahmen eingeleitet, die einen frühzeitigen Beginn der Instruktionsausführung ermöglichen. Adreßtransformation und Überprüfung der Zugriffsrechte spielen hier eine Rolle. So wird ein gleichmäßiger Befehlsstrom mit einer Ausführungsrate von einer Instruktion pro Takt verarbeitet. Die Instruktionen
MOV Register, Register
MOV Memory, Register
MOV Register, Memory
MOV Register, Immediate
ALU Register, Register

(ALU ist eine Arithmetik- oder Logikfunktion, die auf Registeroperanden wirkt, wie beispielsweise Addiere, Subtrahiere, Teste auf Gleichheit etc.)
JCC NOT TAKEN
(bedingte Verzweigung, die nicht zur Ausführung kommt; es wird mit der Instruktion auf das JCC NOT TAKEN fortgefahren)
sind Befehle, die zum Kerninstruktionssatz gehören und deshalb nur einen Takt zu ihrer Ausführung benötigen. Eine bedingte Verzweigung, die den Kontrollfluß des Programms ändert, weil sie zur Ausführung kommt, benötigt drei Takte, da die beiden Instruktionsdekodierstufen der Pipeline geleert werden müssen. Beispiele für Instruktionen, die zwei Takte zu ihrer Ausführung benötigen, sind
ALU Register, Memory
SHIFT Register, Immediate



Zeitlicher Ablauf und Steuerung eines „schnellen" und „langsamen" 486-Burst-Buszyklus

Für die ALU-Instruktion wird ein Takt benötigt, um das Datum aus dem Cache-Speicher zu holen und ein weiterer, um sie auszuführen. Mit einem Extratakt wird der Barrelshifter geladen, bevor die SHIFT-Instruktion in einem weiteren Takt ausgeführt werden kann. Im Durchschnitt benötigt der i486-Prozessor etwa 1,7 bis zwei Takte pro Befehl. Diese Zahlen sind selbstverständlich programmabhängig.

Die Kontrolleinheit

Das Dekodierwerk ist eng mit der Kontrolleinheit des i486-Prozessors verbunden. Mikroinstruktions-Pfade führen sowohl zu den Speicherverwaltungs- und Schutzeinheiten als auch zum eigentlichen Rechenwerk, zur Gleitkommaeinheit und zum Funktionsblock Dekodieren. Die Kontrolleinheit enthält den Mikrocode und den Mikrocodesequenzer. Sie erzeugt die Ausnahmebedingungen, leitet die Interrupts weiter und verarbeitet die Breakpoints. Beide Rechenwerke auf dem Chip teilen das identische Mikrocode-ROM. Ein Bit entscheidet, ob die Instruktion von der Gleitkommaeinheit oder der ALU ausgeführt werden soll. Im eingeschränkten Maße können auch beide gleichzeitig arbeiten. Ist das Zielregister eines Gleitkommabefehls ein Gleitkommaregister, kann parallel zur Ausführung dieser Instruktion die ALU eine andere unabhängige bearbeiten.

Neu auf dem i486-Chip ist die Gleitkommaeinheit, die sich ausschließlich mit Mathematik beschäftigen soll. Auch ihre Verarbeitungsgeschwindigkeit wurde erhöht. Da sich dieses Rechenwerk zudem im Kontext der gesamten Maschine befindet, fallen der Command- und Daten-Port weg, über den der 80387 von der 80386DX-CPU mit Kommandos und Operanden versorgt wurde. Die Gleitkomma-Rechnereinheit auf dem i486-Prozessor ist etwa vier- bis sechsmal schneller als 80387 bei gleicher Taktfrequenz. Das Gleitkomma-Rechenwerk ist selbstverständlich binärkompatibel zu der Kombination 80386DX/80387. Es unterstützt den identischen Instruktionssatz.

Beim Auftreten eines Rechenfehlers legte IBM bei der Systemarchitektur des PC zwischen Prozessor und Koprozessor ein besonderes Fehlerprotokoll fest. Das ERROR-Signal des Koprozessors wurde auf den Interrupt-Eingang 13 des Interrupt-Controllers geschaltet, der seinerseits einen Interrupt-Zyklus initiierte. Der Interrupt-Vektor, der dem Prozessor auf seinen Datenleitungen übergeben wurde, zeigte dann auf eine entsprechende Service-Routine im Betriebssystem, die den Fehler behandelte.

Um auch hier Kompatibilität zu gewährleisten, besitzt der i486-Prozessor einen Pin namens FERR#. Tritt ein Fehler auf und hat der Benutzer über ein neues Bit im CR0-Register IBM-kompatible Behandlung angewählt, kann FERR# auf den

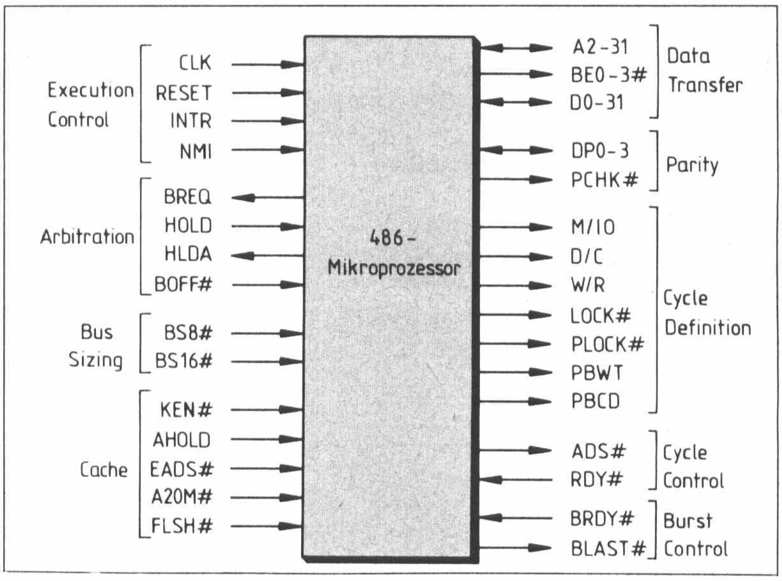
Interrupt-Controller geschaltet werden, der seinerseits wieder den Interrupt-Zyklus initiiert. Mit Hilfe dieses Bits läßt sich aber auch das normale, in der 386-Architektur festgelegte Protokoll anwählen.

Das i486-Programmiermodell und i486-Architekturerweiterungen

Bis auf einen wesentlichen Punkt ist zu dem in der 386-Architektur festgelegten Konzept der Speicherverwaltung und des Speicherschutzes nichts hinzugekommen. Im CR0-Register wurde ein neues Bit – WP = „Write Protect" – definiert (Bit 16). Ist dieses Bit auf 0 gesetzt, entspricht der i486-Prozessor genau der 386-Architektur und arbeitet ohne Unterschied zum 80386DX/SX. Steht es auf 1, wird eine Ausnahmebedingung erzeugt (Exception 14), sobald auf eine Page, die ausschließlich lesenden Zugriff erlaubt, von irgendeiner Privilegstufe aus schreibend zugegriffen wird. Die 386-Architektur erlaubte auch schreibenden Zugriff auf Pages aus den Privilegstufen 0 bis 2, die als „nur lesbar" markiert waren. Nur auf der äußersten Privilegstufe 3, die den Anwendungsprogrammen zugeordnet ist, wurde ein Page-Fault generiert.

Auch der Kern eines Betriebssystems darf nicht mehr auf schreibgeschützte Pages zugreifen, wenn WP = 1 ist. Diese Erweiterung erlaubt darüber hinaus das Betriebssystem-Konzept, das man als „Copy-On-Write" bzw. „Lazy-Copying" bezeichnet. Erzeugt etwa ein Elternprozeß einen neuen, der Kindprozeß genannt wird, so markiert das Betriebssystem alle Pages im logischen oder virtuellen Adreßraum des Kindes als schreibgeschützt. Adressiert das Kind eine solche Page mit schreibendem Zugriff, wird eine Ausnahmebedingung erzeugt, und das Betriebssystem kopiert nur diese Page. Wollte man dem Kind einen eigenen Adreßraum geben, ohne daß es das WP-Bit gäbe, müßte man alle Pages kopieren.

Die neuen Bits 29 = WT und 30 = CE steuern das On-Chip-Cache-System. So ergeben sich folgende mögliche Bit-Kombinationen und ihre Konsequenzen. CE = 0 und WT = 0 ist die zum 80386DX/SX kompatible Form. CE = 1 aktiviert das Cache. Zur Reset-Zeit haben diese Bits den Wert Null, um zum 80386DX/SX kompatibel zu sein. Der Zustand des CE-Bits, der logische Pegel des „Cache Enable Input"-Pins (KEN#) und das aktuelle „Page Cache Disable-(PCD"-Bit) entscheiden, ob eine Cache-Line als Folge eines Cache-Fehlversuchs in das Cache übernommen werden soll oder nicht. CE = 1, KEN# = 0 und PCD = 0) veranlassen die Übernahme. Aufgrund dieser Bedingungen lassen sich bestimmte Pages im System als „Nicht-Cache-fähig" markieren, etwa sol-



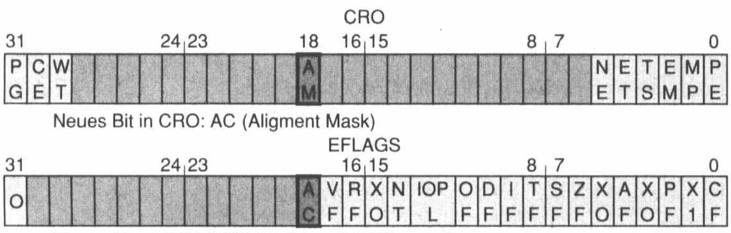
Überblick über die verschiedenen Gruppen der Ein- und Ausgangssignale am 80486

che, die in einem Multiprozessorsystem unter den verschiedenen Prozessoren gemeinsam geteilt werden, oder solche, die zum Eingabe-/Ausgabesystem gehören, oder schließlich die Page-Tabellen selbst. Das WT-Bit („Write Transparent") zeigt an, ob bei einem schreibenden Zugriff des Prozessors die Daten über das Cache auf die Datenleitungen gelangen sollen oder nicht. WT = 1 erlaubt die Invalidierung, also das Ersetzen von Cache-Lines und „Write-Through".

Ist WT = 0, wird ein schreibender Zugriff nicht auf die Pins gesandt, wenn sich die Adresse im Cache befindet, außerdem kann keine Cache-Line durch eine andere ersetzt werden. Auf diese Weise ist der Anwender in der Lage, das Cache als SRAM für Instruktionen und Daten zu verwenden. Am Programmanfang wird er CE und WT auf 1 setzen. Sein Programm ist so aufgebaut, daß jeder Zugriff nur einmal erfolgt, also ständig Cache-Fehlversuche produziert werden und das Cache jedesmal aktualisiert wird. Befindet sich das relevante Programm im Cache, schaltet es der Anwender mit CE = 0 und WT = 0 ab.

Ein weiteres neues Bit ist das „Alignment-Mask", das AM Bit Nummer 18, im CR0-Register. Dieses Bit maskiert das „Alignment-Check"-Bit im EFLAGS-Register, das ebenfalls neu ist. Ist AM = 0,

dann ist AC deaktiviert. Ist AM = 0, dann ist AC aktiviert. AM = 0 verhindert die Auswertung des AC-Bits, so daß selbst 386-Software, die dieses Bit fälschlich verwendet, nicht zur Erzeugung einer Ausnahmebedingung führt. Wurde Alignment Checking aktiviert, prüft die i486-CPU, ob die Daten auf den ihrem Typ entsprechenden Grenzen stehen – ein Doppelwort auf 32-bit-Doppelwortgrenze, ein Wort auf 16-bit-Wortgrenze usw. Ist das nicht der Fall, wird Exception 17 erzeugt. Programmsysteme, die diese Grenzen nicht einhalten, werden langsamer ausgeführt als solche, die es tun, da mehr als nur ein Bus-Zyklus für jeden solchen Fall initiiert werden muß. Nachträglich könnten Systeme dieser Art beschleunigt werden, indem man mittels Alignment Check die Stellen aufspürt und nachträglich mit Hilfe der Assembler-Anweisung ALIGN auf die korrekten Grenzen bringt.



Neues Bit in CRO: AC (Aligment Mask)

Neues Bit im "EFLAGS" Register: AC (Aligment)

AM	AC	CHECK
0	0	NO
0	1	NO
1	0	NO
1	1	YES

Das Alignment-Mask-Bit im Control-Register CR0 und das Alignment-Check-Bit im EFLAGS-Register: In Anwendungen der Künstlichen Intelligenz, KI, erhält der Systementwickler und Benutzer das Tagging-Konzept, welches bei diesen Applikationen häufig verwendet wird, in Hardware. Das Markieren (Tagging) von Objekten in der KI muß nicht mehr in Software gemacht werden. Dadurch ergibt sich eine Leistungssteigerung von ca. 12% in diesem Anwendungsbereich.

Es gibt aber einen zweiten Grund für die Existenz von AC und AM. Programme der Künstlichen Intelligenz benutzen sogenannte Tagbits, um Objekte zu identifizieren, die sie manipulieren. Hat man etwa einen Zeiger auf ein solches Objekt, ist man in der Lage, in den unteren Bits den Typ zu verschlüsseln. Stünde dieser Zeiger auf seiner ihm entsprechenden Grenze, müßten diese Bits = Null sein. Da sie aber manipuliert wurden, wird der i486-Prozessor eine Ausnahmebedingung erzeugen, an deren Ausgang eine Systemroutine steht, die diese Bits entschlüsselt und prüft.

Im Zusammenhang mit dem WP-Bit wurde bereits das PCD-Bit erwähnt. Dieses und das PWT-Bit sind in jedem „Page Table Entry" (PTE) verschlüsselt. Sie befinden sich in beiden Ebenen der Page-Tabellen, und zwar sowohl in den Einträgen in das Page-Inhaltsverzeichnis als auch in den Einträgen der Page-Tabelle. Das PWT-Bit steuert den Schreibmechanismus für die in Arbeit befindliche Page. PWT = 1 bedeutet „Write-Through", PWT = 0 ermöglicht „Write-Back". Da das Cache-System des i486-Prozessors nur „Write-Through" unterstützt, wird dieses Bit ignoriert. Das muß aber nicht so bleiben.

Das PCD-Bit kontrolliert, ob eine Page in das Cache-System übernommen werden darf oder nicht. PCD = 1 bedeutet: Das Betriebssystem will nicht, daß diese Page im Cache installiert wird. Ist PCD = 0, besteht grundsätzlich die Genehmigung dazu. PCD = 0 allein aber genügt nicht. Das CE-Bit muß auf 1 stehen, und auf der KEN#-Leitung muß der logische Pegel 0 vorliegen. Damit die externe Hardware das entsprechende Signal abhängig vom PCD-Bit überhaupt aufschalten kann, ist es auf dem mit ihm assoziierten PCD-Pin außen ablesbar. Das PCD-Bit ist sehr nützlich, da man auf einen ganzen Satz von Abbildungslogik verzichten kann, die man andernfalls in einem System vorsehen muß.

Zusammenfassung

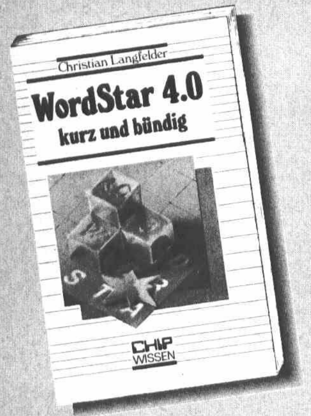
Über die Erklärung des Begriffes Rechnerarchitektur im allgemeinen Sinne wurden die wesentlichen Grundzüge der 386-Architektur beschrieben. Sicherlich steht die neueste Implementation dieser Architektur, der i486-Prozessor, im Vordergrund des Interesses. Es ist aber nützlich, diesen vorerst letzten Schritt in der Kette der 386-Architektur-Revolution vor dem Hintergrund der Architektur zu beleuchten, von der er abstammt.

Impressum

CHIP PLUS erscheint monatlich als Beilage zu CHIP. Anschrift der Redaktion: Schillerstr. 23a, 8000 München 2, Tel. 0 89-5 14 93-0, Verlag: Vogel Verlag und Druck KG, Max-Planck-Str. 7/9, 8700 Würzburg, Tel. 09 31/4 18-0. Redaktionsdirektor: Dr. Gerald O. Dick. Redaktion: Martin Stübs (verantw.), Ulrich Parthier. Grafische Gestaltung: Hans Kuh, Nadja Hanauer. Chef vom Dienst: Reiner Lanninger. Textchef: Dr. Alfred Günter Neumann. Autor dieser Ausgabe: Dr. Jens Bodenkamp. Verlagsdirektor: Dr. Andreas Kaiser. Anzeigen: Friedrich Mangold. Satz, Litho, Druck, Verarbeitung, Versand: Vogel Verlag und Druck KG. Der Verlag haftet nicht für unverlangt eingesandte Manuskripte und Fotos.

CHIP WISSEN

Die kompetente Buchreihe rund um den PC



Christian Langfelder:

WordStar 4.0 kurz und bündig

184 Seiten, 46 Bilder, Hardcover
35,– DM/ISBN 3-8023-**0219**-2

Dieses Buch verfolgt einerseits das bewährte Konzept seines Vorgängers „WordStar kurz und bündig" und macht den Leser andererseits mit den neuen Eigenschaften und Möglichkeiten von WordStar 4.0 bzw. den Zusatzprogrammen von WordStar 4.0 Extra vertraut. Es ist somit für Einsteiger in der Textverarbeitung als auch für Umsteiger auf die neue Version bestens geeignet.



Hans Mahnke:

Strukturiert programmieren in FORTRAN 77

mit Microsoft FORTRAN 4.0

128 Seiten, 20 Bilder, Hardcover
30,– DM/ISBN 3-8023-**0221**-4

Dieses Buch wendet sich an alle, die die Programmiersprache FORTRAN erlernen wollen, also Studenten, Techniker und PC-Benutzer, die gut strukturierte Programme brauchen.
Durch seinen einfachen Aufbau und die Kombination von Programmieren mit der Sprache FORTRAN eignet es sich besonders gut zum Einsatz an Schulen, bei Lehrgängen und Seminaren sowie als Schulungsunterlage in Firmen, wenn dort FORTRAN in Großrechenanlagen programmiert ist.
Aus dem Inhalt:

★ Aufbau von FORTRAN-Programmen
★ COMMON-Bereiche
★ BLOCK-DATA-Unterprogramme
★ EQUIVALENCE-Anweisung
★ Ein- und Ausgaben



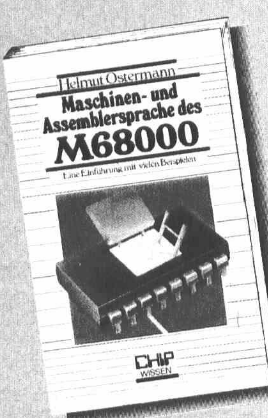
Fred Wagenknecht:

Das 68000er-Animationsbuch

Bewegte Grafik auf Amiga und Atari ST

284 Seiten, 134 Bilder
99 Listings, Hardcover
48,– DM/ISBN 3-8023-**0211**-4

Dieses Buch für Grafikfreaks bietet Designern und Grafikinteressierten einen unkomplizierten Einstieg in die Computeranimation. Anhand relativ einfacher BASIC-Programme wird der Leser Schritt für Schritt in spannende und schöpferische Grafikprogramme eingeführt. Diese 2D- und 3D-Experimente sind überschaubar strukturiert und ausführlich dokumentiert. Der Entwurf der Objekte und der Bewegungsabläufe sind durchschaubar und führen zu sofortigen Erfolgserlebnissen.



Helmut Ostermann:

Maschinen- und Assemblersprache des M68000

Eine Einführung mit vielen Beispielen für Amiga, NDR-Klein-Computer und Sinclair QL

368 Seiten, 62 Bilder
zahlr. Listings, Hardcover
48,– DM/ISBN 3-8023-**0174**-9

Der Mikroprozessor 68000 von Motorola hat durch Computer wie Amiga, Macintosh, Atari St und Sinclair QL in kurzer Zeit eine weite Verbreitung gefunden. Seine erstaunliche Leistungsfähigkeit und die klare Struktur der Adressierungsarten ermöglichen erfolgreiches Programmieren. Angesprochen sind Leser mit Vorkenntnissen in der Programmierung eines Mikroprozessors, während Anfänger mit Hilfe des Glossars und eines Anhangs den Einstieg finden. Die vielen Beispiele wurden auf Amiga, Sinclair QL und NDR-Klein-Computern entwickelt.

Haben Sie schon den neuen „CHIP- Katalog" ?
Bestellen Sie gleich!

VOGEL Buchverlag Würzburg
Postfach 6740 · 8700 Würzburg 1

CHIP WISSEN

Die kompetente Buchreihe rund um den PC

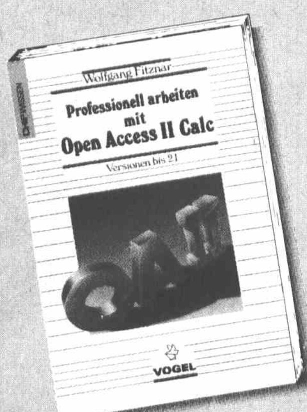


Hans-Ulrich Wolff:

Schneller erfolgreich mit Lotus Manuscript 2.0

192 Seiten, 82 Bilder
Hardcover, Format 24 x 16
40,– DM/ISBN 3-8023-**0222**-2

Dieses Buch gibt eine systematische Einführung in eine leistungsfähige Dokumentationsverarbeitung. Es beschreibt in einer systematischen Einführung die Programmbefehle beispielhaft Schritt für Schritt. Die detaillierte Gliederung sowie das umfangreiche Stichwortverzeichnis ermöglichen dem engagierten und geübteren Anwender das rasche Nachschlagen der ihn speziell interessierenden Abschnitte. Das Erzeugen hochwertiger Ausdrucke mit Grafiken, Tabellen und Gleichungen macht seinen Einsatz vor allem für technisch-wissenschaftliche Anwendungen interessant.

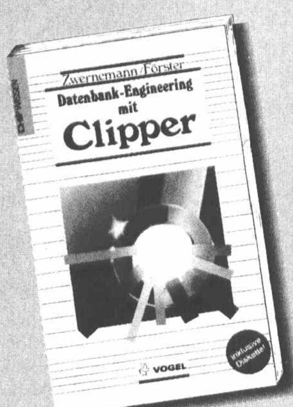


Wolfgang Fitznar:

Professionell arbeiten mit Open Access II Calc

Versionen bis 2.1
340 Seiten, 103 Bilder, 89 Tabellen
Hardcover, Format 24 x 16
48,– DM/ISBN 3-8023-**0298**-2

Dieses Buch unterstützt und begleitet den Anwender beim Einarbeiten in das Kalkulationsmodul »Calc« und die allgemeinen Bestandteile von Open Access II unter Berücksichtigung der neuen Version 2.1.
Aus dem Inhalt:
★ viele leicht nachvollziehbare und praxisnahe Beispiele ★ beigelegte Tastaturschablonen ★ aufwendige Grafiken ★ fundierte Besprechungen zu Kalkulationsmakros.
Der durchdachte Aufbau macht das Buch in Schulungen wie zum Selbststudium einsetzbar.

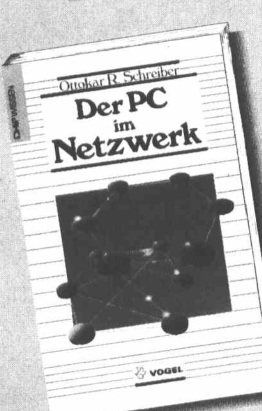


Zwernemann/Förster:

Datenbank-Engineering mit Clipper

464 Seiten, 58 Bilder, Hardcover
68,– DM **(inkl. 5,25-Zoll-Diskette)**
Format 24 x 16, ISBN 3-8023-**0263**-X

Dieses Buch beschreibt die Engineering-Praxis mit Clipper, welches **das** Entwicklungsprogramm für Datenbanksysteme ist. Der Inhalt konzentriert sich ausschließlich auf die Programmplanung, Entwicklung und Realisierung, wie sie im Rahmen eines kleinen Entwicklungsteams oder eines einzelnen Programmierers erfolgen sollte.
Typische Anwender sind mittelständische Unternehmen und Kleinbetriebe. Hierfür wurde ein konkretes Praxisbeispiel (ein komplettes Adreßverwaltungsprogramm) erarbeitet. Auf der beigefügten **5,25-Zoll-Diskette** finden Sie das komplette Datenbankprogramm, Checkliste und Planungsunterlagen.



Ottokar R. Schreiber:

Der PC im Netzwerk

224 Seiten, 72 Bilder, 18 Tabellen
Hardcover, Format 24 x 16
40,– DM/ISBN 3-8023-**0262**-1

Dieses Buch bietet fachliches Grundwissen über die Vernetzung und stellt schwierige Sachverhalte verständlich dar. Umfassende Informationssysteme, zunächst als Insellösungen in Form lokaler Netzwerke, halten Einzug in Büros, Unis usw. Es entsteht die Forderung über Verbindung dieser Kommunikationsinseln über Gateways, mit Großrechnern sowie Daten- und Wissensbanken. Zahlreiche LANs verbinden sich zu einem Internet, in dem jeder Teilnehmer auf alle darin verfügbaren Ressourcen zugreifen kann. Also insgesamt ein interessantes Gebiet. Greifen Sie zu!!

VOGEL

Haben Sie schon den neuen „CHIP-Katalog"?
Bestellen Sie gleich!

Vogel Buchverlag
Postfach 67 40 · 8700 Würzburg 1